sun®
microsystems

SunOS Reference Manual

Part Number: 800-3827-10
Revision A of 27 March, 1990

NAME

intro – introduction to system services and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

This section describes all of the system calls.

A 2V section number means one or more of the following:

- The man page documents System V behavior only.
- The man page documents default SunOS behavior and System V behavior as it differs from the default behavior. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents behavior compliant with *IEEE Std 1003.1-1988* (POSIX.1).

Compile programs for the System V environment using **/usr/5bin/cc**. Compile programs for the default SunOS environment using **/usr/bin/cc**. The following man pages describe the various environments provided by Sun: **lint**(1V), **ansic**(7V), **bsd**(7), **posix**(7V), **sunos**(7V), **svidii**(7V), **svidiii**(7V), **xopen**(7V).

Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible return value. This is almost always '**–1**'; the individual descriptions specify the details. An error code is also made available in the external variable **errno**. **errno** is not cleared on successful calls, so it should be tested only after an error has been indicated. Note: several system calls overload the meanings of these error numbers, and the meanings must be interpreted according to the type and circumstances of the call. See **ERROR CODES** below for a list of system error codes.

As with normal arguments, all return codes and values from functions are of type integer unless otherwise noted.

The rest of this man page is organized as follows:

SYSTEM PARAMETERS	System limits, values and options.
DEFINITIONS	System abstractions and services.
STREAMS	Modular communication between software layers (tty system, networking).
SYSTEM V IPC	System V shared memory, semaphores, and messages.
ERROR CODES	A list of system error codes with descriptions.
LIST OF SYSTEM CALLS	A list of all system calls with brief descriptions.

SYSTEM PARAMETERS

Sections 2 and 3 support a naming convention for those system parameters that may change from one object to another (for example, path name length may is 255 on a UFS file system but may be 14 on an NFS file system exported by a System V based server). Typically, the system has to be queried (using **pathconf**(2V), **fpathconf**( ), or **sysconf**(2V)) to retrieve the parameter of interest. The parameters have conceptual names such as PATH_MAX. These names are defined in header files if and only if they are invariant across all file systems and releases of the operating system, that is, very rarely. Because they *may* be defined and/or available from the system calls, there have to be separate names for the parameters and their values. The notation {PATH_MAX} denotes the value of the parameter PATH_MAX. Do not confuse this with **_PC_PATH_MAX**, the name that is passed to the system call to retrieve the value:

```
maxpathlen = pathconf(".", _PC_PATH_MAX);
```

See **pathconf**(2V), and **sysconf**(2V) for further information about these parameters.

DEFINITIONS

Controlling Terminal

A terminal that is associated with a session. Each session may have at most one controlling terminal; a terminal may be the controlling terminal of at most one session. The controlling terminal is used to direct signals (such as interrupts and job control signals) to the appropriate processes by way of the tty's process group. Controlling terminals are assigned when a session leader opens a terminal file that is not currently a controlling terminal.

Descriptor

An integer assigned by the system when a file is referenced by **open**(2V), **dup**(2V), or **pipe**(2V) or a socket is referenced by **socket**(2) or **socketpair**(2) that uniquely identifies an access path to that file or socket from a given process or any of its children.

Directory

A directory is a special type of file that contains entries that are references to other files. Directory entries are called links. By convention, a directory contains at least two links, '.' and '..', referred to as *dot* and *dot-dot* respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

Effective User ID, Effective Group ID, and Access Groups

Access to system resources is governed by three values: the effective user ID, the effective group ID, and the supplementary group ID.

The effective user ID and effective group ID are initially the process's real user ID and real group ID respectively. Either may be modified through execution of a set-user-ID or set-group-ID file (possibly by one of its ancestors) (see **execve**(2V)).

The supplementary group ID are an additional set of group ID's used only in determining resource accessibility. Access checks are performed as described below in **File Access Permissions.**

File Access Permissions

Every file in the file system has a set of access permissions. These permissions are used in determining whether a process may perform a requested operation on the file (such as opening a file for writing). Access permissions are established at the time a file is created. They may be changed at some later time through the **chmod**(2V) call.

File access is broken down according to whether a file may be: read, written, or executed. Directory files use the execute permission to control if the directory may be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the owner of the file, those users in the file's group, anyone else. Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if:

> The process's effective user ID is that of the super-user.
>
> The process's effective user ID matches the user ID of the owner of the file and the owner permissions allow the access.
>
> The process's effective user ID does not match the user ID of the owner of the file, and either the process's effective group ID matches the group ID of the file, or the group ID of the file is in the process's supplementary group IDs, and the group permissions allow the access.
>
> Neither the effective user ID nor effective group ID and supplementary group IDs of the process match the corresponding user ID and group ID of the file, but the permissions for "other users" allow access.

Otherwise, permission is denied.

File Name

Names consisting of up to {NAME_MAX} characters may be used to name an ordinary file, special file, or directory.

These characters may be selected from the set of all ASCII character excluding \0 (null) and the ASCII code for / (slash). (The parity bit, bit 8, must be 0.)

Note: it is generally unwise to use *, ?, [, or] as part of file names because of the special meaning attached to these characters by the shell. See **sh**(1). Although permitted, it is advisable to avoid the use of unprintable characters in file names.

Parent Process ID

A new process is created by a currently active process **fork (2V).** The parent process ID of a process is the process ID of its creator.

Path Name and Path Prefix

A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name. The total length of a path name must be less than **{PATH_MAX}** characters.

More precisely, a path name is a null-terminated character string constructed as follows:

<path-name>::=*<file-name>*| *<path-prefix><file-name>*| /
<path-prefix>::=*<rtprefix>*| /*<rtprefix>*
<rtprefix>::=*<dirname>*/| *<rtprefix><dirname>*/

where *<file-name>* is a string of 1 to **{NAME_MAX}** characters other than the ASCII slash and null, and *<dirname>* is a string of 1 to **{NAME_MAX}** characters (other than the ASCII slash and null) that names a directory.

If a path name begins with a slash, the search begins at the *root* directory. Otherwise, the search begins at the current working directory.

A slash, by itself, names the root directory. A dot (**.**) names the current working directory.

A null path name also refers to the current directory. However, this is not true of all UNIX systems. (On such systems, accidental use of a null path name in routines that do not check for it may corrupt the current working directory.) For portable code, specify the current directory explicitly using '"."', rather than '""'.

Process Group ID

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes (see the description of **killpg**( ) on **kill**(2V)) and the job control mechanisms of **csh**(1). Process groups exist from their creation until the last member is reaped (that is, a parent issued a call to **wait**(2V)).

Process ID

Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to MAXPID (see **<sys/param.h>**).

Real User ID and Real Group ID

Each user on the system is identified by a positive integer termed the real user ID.

Each user is also a member of one or more groups. One of these groups is distinguished from others and used in implementing accounting facilities. The positive integer corresponding to this distinguished group is termed the real group ID.

All processes have a real user ID and real group ID. These are initialized from the equivalent attributes of the process that created it.

Root Directory and Current Working Directory

Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. The root directory is used as the starting point for absolute path name resolution. The current working directory is used as the starting point for relative path name resolution. A process's root directory need not be (but typically is) the root directory of the root file system.

Session

Each process is a member of a session. A session is associated with each controlling terminal in the system, such as login shells and windows. Each process is created in the session of its parent. A process may alter its session using **setsid**(2V) if it is not already a session leader. The system supports session IDs. A session leader is a process having process ID equal to process group ID equal to session ID. Only a session leader may acquire a controlling terminal. In SunOS Release 4.1, processes are created in sessions by **init**(8) and **inetd (8C)**. Sessions are also created for processes that disassociate themselves from a controlling terminal using

ioctl(fd, TIOCNOTTY, 0)

or

setpgrp(mypid, 0) For more information about sessions, see **setsid**(2V).

Signal

Signals are used for notification of asynchronous events. Signals may directed to processes, process groups, and other combinations of processes. Signals may be sent by a process or by the operating system. Some signals may be caught. There is typically a default behavior on receipt if they are not caught. For more information about signals, see **signal**(3V), **kill**(2V), **sigvec**(2), **termio**(4).

Sockets and Address Families

A socket is an endpoint for communication between processes, similar to the way a telephone is the endpoint of communication between humans. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties. These properties include whether messages sent and received at a socket require the name of the partner, whether communication is reliable, the format used in naming message recipients, etc.

Each instance of the system supports some collection of socket types; consult **socket**(2) for more information about the types available and their properties.

Each instance of the system supports some number of sets of communications protocols. Each protocol set supports addresses of a certain format. An Address Family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

Special Processes

The processes with a process ID's of 0, 1, and 2 are special. Process 0 is the scheduler. Process 1 is the initialization process **init**, and is the ancestor of every other process in the system. It is used to control the process structure. Process 2 is the paging daemon.

Super-user

A process is recognized as a *super-user* process and is granted special privileges if its effective user ID is 0.

Tty Process Group

Each active process can be a member of a terminal group that is identified by a positive integer called the tty process group ID. This grouping is used to arbitrate between multiple jobs contending for the same terminal (see **csh**(1), and **termio**(4)), to direct signals (tty and job control) to the appropriate process group, and to terminate a group of related processes upon termination of one of the processes in the group (see **exit**(2V) and **sigvec**(2)).

STREAMS

A set of kernel mechanisms that support the development of network services and data communication *drivers*. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures.

Stream

A stream is a full-duplex data path within the kernel between a user process and driver routines. The primary components are a stream head, a *driver* and zero or more *modules* between the stream head and *driver*. A stream is analogous to a Shell pipeline except that data flow and processing are bidirectional.

Stream Head

In a stream, the stream head is the end of the stream that provides the interface between the stream and a user process. The principle functions of the stream head are processing STREAMS-related system calls, and passing data and information between a user process and the stream.

Driver

In a stream, the *driver* provides the interface between peripheral hardware and the stream. A *driver* can also be a pseudo-*driver*, such as a *multiplexor* or *emulator*, and need not be associated with a hardware device.

Module

A module is an entity containing processing routines for input and output data. It always exists in the middle of a stream, between the stream's head and a *driver*. A *module* is the STREAMS counterpart to the commands in a Shell pipeline except that a module contains a pair of functions which allow independent bidirectional (*downstream* and *upstream*) data flow and processing.

Downstream

In a stream, the direction from stream head to *driver*.

Upstream

In a stream, the direction from *driver* to stream head.

Message

In a stream, one or more blocks of data or information, with associated STREAMS control structures. Messages can be of several defined types, which identify the message contents. Messages are the only means of transferring data and communicating within a stream.

Message Queue

In a stream, a linked list of *messages* awaiting processing by a *module* or *driver*.

Read Queue

In a stream, the *message queue* in a *module* or *driver* containing *messages* moving *upstream*.

Write Queue

In a stream, the *message queue* in a *module* or *driver* containing *messages* moving *downstream*.

Multiplexor

A multiplexor is a driver that allows STREAMS associated with several user processes to be connected to a single *driver*, or several *drivers* to be connected to a single user process. STREAMS does not provide a general multiplexing *driver*, but does provide the facilities for constructing them, and for connecting multiplexed configurations of STREAMS.

SYSTEM V IPC

The SunOS system supports the System V IPC namespace. For information about shared memory, semaphores and messages see **msgctl**(2), **msgget**(2), **msgop**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2) and **shmop**(2).

ERROR CODES

Each system call description attempts to list all possible error numbers. The following is a complete list of the error numbers and their names as given in **<errno.h>**.

E2BIG 7 Arg list too long
: An argument list longer than 1,048,576 bytes is presented to **execve**(2V) or a routine that called **execve( )**.

EACCES 13 Permission denied
: An attempt was made to access a file in a way forbidden by the protection system.

EADDRINUSE 48 Address already in use
: Only one usage of each address is normally permitted.

EADDRNOTAVAIL 49 Can't assign requested address
: Normally results from an attempt to create a socket with an address not on this machine.

EADV 83 Advertise error
: An attempt was made to advertise a resource which has been advertised already, or to stop the RFS while there are resources still advertised, or to force unmount a resource when it is still advertised. This error is RFS specific.

EAFNOSUPPORT 47 Address family not supported by protocol family
: An address incompatible with the requested protocol was used. For example, you should not necessarily expect to be able to use PUP Internet addresses with ARPA Internet protocols.

EAGAIN 11 No more processes
: A **fork**(2V) failed because the system's process table is full or the user is not allowed to create any more processes, or a system call failed because of insufficient resources.

EALREADY 37 Operation already in progress
: An operation was attempted on a non-blocking object that already had an operation in progress.

EBADF 9 Bad file number
: Either a file descriptor refers to no open file, or a read (respectively, write) request is made to a file that is open only for writing (respectively, reading).

EBADMSG 76 Not a data message
: During a **read**(2V), **getmsg**(2), or **ioctl**(2) I_RECVFD system call to a STREAMS device, something has come to the head of the queue that cannot be processed. That something depends on the system call

read(2V)	control information or a passed file descriptor.
getmsg(2)	passed file descriptor.
ioctl(2)	control or data information.

EBUSY 16 Device busy
: An attempt was made to mount a file system that was already mounted or an attempt was made to dismount a file system on which there is an active file (open file, mapped file, current directory, or mounted-on directory).

ECHILD 10 No children
: A **wait**(2V) was executed by a process that had no existing or unwaited-for child processes.

ECOMM 85 Communication error on send
: An attempt was made to send messages to a remote machine when no virtual circuit could be found. This error is RFS specific.

ECONNABORTED 53 Software caused connection abort
: A connection abort was caused internal to your host machine.

ECONNREFUSED 61 Connection refused
: No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.

ECONNRESET 54 Connection reset by peer
A connection was forcibly closed by a peer. This normally results from the peer executing a **shutdown**(2) call.

EDEADLK 78 Deadlock situation detected/avoided
An attempt was made to lock a system resource that would have resulted in a deadlock situation.

EDESTADDRREQ 39 Destination address required
A required address was omitted from an operation on a socket.

EDOM 33 Math argument
The argument of a function in the math library (as described in section 3M) is out of the domain of the function.

EDQUOT 69 Disc quota exceeded
A **write**( ) to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because the user's quota of disk blocks was exhausted, or the allocation of an inode for a newly created file failed because the user's quota of inodes was exhausted.

EEXIST 17 File exists
An existing file was mentioned in an inappropriate context, for example, **link**(2V).

EFAULT 14 Bad address
The system encountered a hardware fault in attempting to access the arguments of a system call.

EFBIG 27 File too large
The size of a file exceeded the maximum file size (1,082,201,088 bytes).

EHOSTDOWN 64 Host is down
A socket operation failed because the destination host was down.

EHOSTUNREACH 65 Host is unreachable
A socket operation was attempted to an unreachable host.

EIDRM 77 Identifier removed
This error is returned to processes that resume execution due to the removal of an identifier.

EINPROGRESS 36 Operation now in progress
An operation that takes a long time to complete (such as a **connect**(2)) was attempted on a non-blocking object (see **ioctl**(2)).

EINTR 4 Interrupted system call
An asynchronous signal (such as interrupt or quit) that the process has elected to catch occurred during a system call. If execution is resumed after processing the signal, and the system call is not restarted, it will appear as if the interrupted system call returned this error condition.

EINVAL 22 Invalid argument
A system call was made with an invalid argument; for example, dismounting a non-mounted file system, mentioning an unknown signal in **sigvec**( ) or **kill**( ), reading or writing a file for which **lseek**( ) has generated a negative pointer, or some other argument inappropriate for the call. Also set by math functions, see **intro**(3).

EIO 5 I/O error
Some physical I/O error occurred. This error may in some cases occur on a call following the one to which it actually applies.

EISCONN 56 Socket is already connected
A **connect**( ) request was made on an already connected socket; or, a **sendto**( ) or **sendmsg**( ) request on a connected socket specified a destination other than the connected party.

EISDIR 21 Is a directory
An attempt was made to write on a directory.

EISDIR 21 Is a directory
An attempt was made to write on a directory.

ELOOP 62 Too many levels of symbolic links
A path name lookup involved more than 20 symbolic links.

EMFILE 24 Too many open files
A process tried to have more open files than the system allows a process to have. The customary configuration limit is 64 per process.

EMLINK 31 Too many links
An attempt was made to make more than 32767 hard links to a file.

EMSGSIZE 40 Message too long
A message sent on a socket was larger than the internal message buffer.

EMULTIHOP 87 Multihop attempted
An attempt was made to access remote resources which are not directly accessible. This error is RFS specific.

ENAMETOOLONG 63 File name too long
A component of a path name exceeded 255 characters, or an entire path name exceeded 1024 characters.

ENETDOWN 50 Network is down
A socket operation encountered a dead network.

ENETRESET 52 Network dropped connection on reset
The host you were connected to crashed and rebooted.

ENETUNREACH 51 Network is unreachable
A socket operation was attempted to an unreachable network.

ENFILE 23 File table overflow
The system's table of open files is full, and temporarily no more **open**( ) calls can be accepted.

ENOBUFS 55 No buffer space available
An operation on a socket or pipe was not performed because the system lacked sufficient buffer space.

ENODEV 19 No such device
An attempt was made to apply an inappropriate system call to a device (for example, an attempt to read a write-only device) or an attempt was made to use a device not configured by the system.

ENOENT 2 No such file or directory
This error occurs when a file name is specified and the file should exist but does not, or when one of the directories in a path name does not exist.

ENOEXEC 8 Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see **a.out**(5)).

ENOLCK 79 No locks available
A system-imposed limit on the number of simultaneous file and record locks was reached and no more were available at that time.

ENOLINK 82 Link has be severed
The link (virtual circuit) connecting to a remote machine is gone. This error is RFS specific.

ENOMEM 12 Not enough memory

During an **execve**(2V), **sbrk**( ), or **brk**(2), a program asks for more address space or swap space than the system is able to supply, or a process size limit would be exceeded. A lack of swap space is normally a temporary condition; however, a lack of address space is not a temporary condition. The maximum size of the text, data, and stack segments is a system parameter. Soft limits may be increased to their corresponding hard limits.

ENOMSG 75 No message of desired type

An attempt was made to receive a message of a type that does not exist on the specified message queue; see **msgop**(2).

ENONET 80 Machine is not on the network

A attempt was made to advertise, unadvertise, mount, or unmount remote resources while the machine has not done the proper startup to connect to the network. This error is Remote File Sharing (RFS) specific.

ENOPROTOOPT 42 Option not supported by protocol

A bad option was specified in a **setsockopt**( ) or **getsockopt**(2) call.

ENOSPC 28 No space left on device

A **write**( ) to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because no more disk blocks are available on the file system, or the allocation of an inode for a newly created file failed because no more inodes are available on the file system.

ENOSR 74 Out of stream resources

During a STREAMS **open**(2V), either no STREAMS queues or no STREAMS head data structures were available.

ENOSTR 72 Not a stream device

A **putmsg**(2) or **getmsg**(2) system call was attempted on a file descriptor that is not a STREAMS device.

ENOSYS 90 Function not implemented

An attempt was made to use a function that is not available in this implementation.

ENOTBLK 15 Block device required

A file that is not a block device was mentioned where a block device was required, for example, in **mount**(2V).

ENOTCONN 57 Socket is not connected

An request to send or receive data was disallowed because the socket is not connected.

ENOTDIR 20 Not a directory

A non-directory was specified where a directory is required, for example, in a path prefix or as an argument to **chdir**(2V).

ENOTEMPTY 66 Directory not empty

An attempt was made to remove a directory with entries other than '**&.**' and '**&.|.**' by performing a **rmdir**( ) system call or a **rename**( ) system call with that directory specified as the target directory.

ENOTSOCK 38 Socket operation on non-socket

Self-explanatory.

ENOTTY 25 Inappropriate ioctl for device

The code used in an **ioctl**( ) call is not supported by the object that the file descriptor in the call refers to.

ENXIO 6 No such device or address

I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

EOPNOTSUPP 45 Operation not supported on socket
For example, trying to *accept* a connection on a datagram socket.

EPERM 1 Not owner
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

EPFNOSUPPORT 46 Protocol family not supported
The protocol family has not been configured into the system or no implementation for it exists.

EPIPE 32 Broken pipe
An attempt was made to write on a pipe or socket for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is caught or ignored.

EPROTO 86 Protocol error
Some protocol error occurred. This error is device specific, but is generally not related to a hardware failure.

EPROTONOSUPPORT 43 Protocol not supported
The protocol has not been configured into the system or no implementation for it exists.

EPROTOTYPE 41 Protocol wrong type for socket
A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type **SOCK_STREAM**.

ERANGE 34 Result too large
The value of a function in the math library (as described in section 3M) is unrepresentable within machine precision.

EREMOTE 71 Too many levels of remote in path
An attempt was made to remotely mount a file system into a path that already has a remotely mounted component.

EROFS 30 Read-only file system
An attempt to modify a file or directory was made on a file system mounted read-only.

ERREMOTE 81 Object is remote
An attempte was made to advertise a resource which is not on the local machine, or to mount/unmount a device (or pathname) that is on a remote machine. This error is RFS specific.

ESHUTDOWN 58 Can't send after socket shutdown
A request to send data was disallowed because the socket had already been shut down with a previous **shutdown**(2) call.

ESOCKTNOSUPPORT 44 Socket type not supported
The support for the socket type has not been configured into the system or no implementation for it exists.

ESPIPE 29 Illegal seek
An **lseek( )** was issued to a socket or pipe. This error may also be issued for other non-seekable devices.

ESRCH 3 No such process
The process or process group whose number was given does not exist, or any such process is already dead.

ESRMNT 84 Srmount error
An attempt was made to stop RFS while there are resources still mounted by remote machines. This error is RFS specific.

ESTALE 70 Stale NFS file handle
An NFS client referenced a file that it had opened but that had since been deleted.

ETIME 73 Timer expired
The timer set for a STREAMS **ioctl**(2) call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the **ioctl**(2) operation is indeterminate.

ETIMEDOUT 60 Connection timed out
A *connect* request or an NFS request failed because the party to which the request was made did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.)

ETXTBSY 26 Text file busy
An attempt was made to execute a pure-procedure program that is currently open for writing, or an attempt was made to open for writing a pure-procedure program that is being executed.

EUSERS 68 Too many users
An operation to read disk quota information for the user failed because the system quota table was full.

EWOULDBLOCK 35 Operation would block
An operation that would cause a process to block was attempted on an object in non-blocking mode (see **ioctl**(2)).

EXDEV 18 Cross-device link
A hard link to a file on another file system was attempted.

unused 0

SEE ALSO

brk(2), **chdir**(2V), **chmod**(2V), **connect**(2), **dup**(2V), **execve**(2V), **exit**(2V), **fork**(2V), **getmsg**(2), **getsockopt**(2), **ioctl**(2), **killpg**(2), **link**(2V), **mount**(2V), **msgctl**(2), **msgget**(2), **msgop**(2), **open**(2V), **pipe**(2V), **putmsg**(2), **read**(2V), **semctl**(2), **semget**(2), **semop**(2), **getsockopt**(2), **shmctl**(2), **shmget**(2), **shmop**(2), **shutdown**(2), **sigvec**(2), **socket**(2), **socketpair**(2), **wait**(2V), **csh**(1), **sh**(1), **intro(3)**, **perror(3)** **termio**(4), **a.out**(5)

LIST OF SYSTEM CALLS

Name Appears on Page Description

accept	**accept**(2)	accept a connection on a socket
access	**access**(2V)	determine accessibility of file
acct	**acct**(2V)	turn accounting on or off
adjtime	**adjtime**(2)	correct the time to allow synchronization of the system clock
async_daemon	**nfssvc**(2)	NFS daemons
audit	**audit**(2)	write a record to the audit log
auditon	**auditon**(2)	manipulate auditing
auditsvc	**auditsvc**(2)	write audit records to specified file descriptor
bind	**bind**(2)	bind a name to a socket
brk	**brk**(2)	change data segment size
chdir	**chdir**(2V)	change current working directory
chmod	**chmod**(2V)	change mode of file
chown	**chown**(2V)	change owner and group of a file
chroot	**chroot**(2)	change root directory
close	**close**(2V)	delete a descriptor
connect	**connect**(2)	initiate a connection on a socket
creat	**creat**(2V)	create a new file
dup	**dup**(2V)	duplicate a descriptor
dup2	**dup**(2V)	duplicate a descriptor

execve	**execve**(2V)	execute a file
_exit	**exit**(2V)	terminate a process
fchmod	**chmod**(2V)	change mode of file
fchown	**chown**(2V)	change owner and group of a file
fcntl	**fcntl**(2V)	file control
flock	**flock**(2)	apply or remove an advisory lock on an open file
fork	**fork**(2V)	create a new process
fpathconf	**pathconf**(2V)	query file system related limits and options
fstat	**stat**(2V)	get file status
fstatfs	**statfs**(2)	get file system statistics
fsync	**fsync**(2)	synchronize a file's in-core state with that on disk
ftruncate	**truncate**(2)	set a file to a specified length
getauid	**getauid**(2)	get and set user audit identity
getdents	**getdents**(2)	gets directory entries in a filesystem independent format
getdirentries	**getdirentries**(2)	gets directory entries in a filesystem independent format
getdomainname	**getdomainname**(2)	get/set name of current domain
getdtablesize	**getdtablesize**(2)	get descriptor table size
getegid	**getgid**(2V)	get group identity
geteuid	**getuid**(2V)	get user identity
getgid	**getgid**(2V)	get group identity
getgroups	**getgroups**(2V)	get or set supplementary group IDs
gethostid	**gethostid**(2)	get unique identifier of current host
gethostname	**gethostname**(2)	get/set name of current host
getitimer	**getitimer**(2)	get/set value of interval timer
getmsg	**getmsg**(2)	get next message from a stream
getpagesize	**getpagesize**(2)	get system page size
getpeername	**getpeername**(2)	get name of connected peer
getpgrp	**getpgrp**(2V)	return or set the process group of a process
getpid	**getpid**(2V)	get process identification
getppid	**getpid**(2V)	get process identification
getpriority	**getpriority**(2)	get/set process nice value
getrlimit	**getrlimit**(2)	control maximum system resource consumption
getrusage	**getrusage**(2)	get information about resource utilization
getsockname	**getsockname**(2)	get socket name
getsockopt	**getsockopt**(2)	get and set options on sockets
gettimeofday	**gettimeofday**(2)	get or set the date and time
getuid	**getuid**(2V)	get user identity
ioctl	**ioctl**(2)	control device
kill	**kill**(2V)	send a signal to a process or a group of processes
killpg	**killpg**(2)	send signal to a process group
link	**link**(2V)	make a hard link to a file
listen	**listen**(2)	listen for connections on a socket
lseek	**lseek**(2V)	move read/write pointer
lstat	**stat**(2V)	get file status
mctl	**mctl**(2)	memory management control
mincore	**mincore**(2)	determine residency of memory pages
mkdir	**mkdir**(2V)	make a directory file
mkfifo	**mknod**(2V)	make a special file
mknod	**mknod**(2V)	make a special file
mmap	**mmap**(2)	map pages of memory
mount	**mount**(2V)	mount file system
mprotect	**mprotect**(2)	set protection of memory mapping
msgctl	**msgctl**(2)	message control operations

msgget	**msgget**(2)	get message queue
msgop	**msgop**(2)	message operations
msgrcv	**msgop**(2)	message operations
msgsnd	**msgop**(2)	message operations
msync	**msync**(2)	synchronize memory with physical storage
munmap	**munmap**(2)	unmap pages of memory.
nfssvc	**nfssvc**(2)	NFS daemons
open	**open**(2V)	open or create a file for reading or writing
pathconf	**pathconf**(2V)	query file system related limits and options
pipe	**pipe**(2V)	create an interprocess communication channel
poll	**poll**(2)	I/O multiplexing
profil	**profil**(2)	execution time profile
ptrace	**ptrace**(2)	process trace
putmsg	**putmsg**(2)	send a message on a stream
quotactl	**quotactl**(2)	manipulate disk quotas
read	**read**(2V)	read input
readlink	**readlink**(2)	read value of a symbolic link
readv	**read**(2V)	read input
reboot	**reboot**(2)	reboot system or halt processor
recv	**recv**(2)	receive a message from a socket
recvfrom	**recv**(2)	receive a message from a socket
recvmsg	**recv**(2)	receive a message from a socket
rename	**rename**(2V)	change the name of a file
rmdir	**rmdir**(2V)	remove a directory file
sbrk	**brk**(2)	change data segment size
select	**select**(2)	synchronous I/O multiplexing
semctl	**semctl**(2)	semaphore control operations
semget	**semget**(2)	get set of semaphores
semop	**semop**(2)	semaphore operations
send	**send**(2)	send a message from a socket
sendmsg	**send**(2)	send a message from a socket
sendto	**send**(2)	send a message from a socket
setaudit	**setuseraudit**(2)	set the audit classes for a specified user ID
setauid	**getauid**(2)	get and set user audit identity
setdomainname	**getdomainname**(2)	get/set name of current domain
setgroups	**getgroups**(2V)	get or set supplementary group IDs
sethostname	**gethostname**(2)	get/set name of current host
setitimer	**getitimer**(2)	get/set value of interval timer
setpgid	**setpgid**(2V)	set process group ID for job control
setpgrp	**getpgrp**(2V)	return or set the process group of a process
setpriority	**getpriority**(2)	get/set process nice value
setregid	**setregid**(2)	set real and effective group IDs
setreuid	**setreuid**(2)	set real and effective user IDs
setrlimit	**getrlimit**(2)	control maximum system resource consumption
setsid	**setsid**(2V)	create session and set process group ID
setsockopt	**getsockopt**(2)	get and set options on sockets
settimeofday	**gettimeofday**(2)	get or set the date and time
setuseraudit	**setuseraudit**(2)	set the audit classes for a specified user ID
sgetl	**sputl**(2)	access long integer data in a machine-independent fashion
shmat	**shmop**(2)	shared memory operations
shmctl	**shmctl**(2)	shared memory control operations
shmdt	**shmop**(2)	shared memory operations
shmget	**shmget**(2)	get shared memory segment identifier

shmop	**shmop**(2)	shared memory operations
shutdown	**shutdown**(2)	shut down part of a full-duplex connection
sigblock	**sigblock**(2)	block signals
sigmask	**sigblock**(2)	block signals
sigpause	**sigpause**(2V)	automatically release blocked signals and wait for interrupt
sigpending	**sigpending**(2V)	examine pending signals
sigprocmask	**sigprocmask**(2V)	examine and change blocked signals
sigsetmask	**sigsetmask**(2)	set current signal mask
sigstack	**sigstack**(2)	set and/or get signal stack context
sigsuspend	**sigpause**(2V)	automatically release blocked signals and wait for interrupt
sigvec	**sigvec**(2)	software signal facilities
socket	**socket**(2)	create an endpoint for communication
socketpair	**socketpair**(2)	create a pair of connected sockets
sputl	**sputl**(2)	access long integer data in a machine-independent fashion
stat	**stat**(2V)	get file status
statfs	**statfs**(2)	get file system statistics
swapon	**swapon**(2)	add a swap device for interleaved paging/swapping
symlink	**symlink**(2)	make symbolic link to a file
sync	**sync**(2)	update super-block
syscall	**syscall**(2)	indirect system call
sysconf	**sysconf**(2V)	query system related limits, values, options
tell	**lseek**(2V)	move read/write pointer
truncate	**truncate**(2)	set a file to a specified length
umask	**umask**(2V)	set file creation mode mask
umount	**unmount**(2V)	remove a file system
uname	**uname**(2V)	get information about current system
unlink	**unlink**(2V)	remove directory entry
unmount	**unmount**(2V)	remove a file system
ustat	**ustat**(2)	get file system statistics
utimes	**utimes**(2)	set file times
vadvise	**vadvise**(2)	give advice to paging system
vfork	**vfork**(2)	spawn new process in a virtual memory efficient way
vhangup	**vhangup**(2)	virtually ''hangup'' the current control terminal
wait	**wait**(2V)	wait for process to terminate or stop, examine returned statu
wait3	**wait**(2V)	wait for process to terminate or stop, examine returned statu
wait4	**wait**(2V)	wait for process to terminate or stop, examine returned statu
waitpid	**wait**(2V)	wait for process to terminate or stop, examine returned statu
WEXITSTATUS	**wait**(2V)	wait for process to terminate or stop, examine returned statu
WIFEXITED	**wait**(2V)	wait for process to terminate or stop, examine returned statu
WIFSIGNALED	**wait**(2V)	wait for process to terminate or stop, examine returned statu
WIFSTOPPED	**wait**(2V)	wait for process to terminate or stop, examine returned statu
write	**write**(2V)	write output
writev	**write**(2V)	write output
WSTOPSIG	**wait**(2V)	wait for process to terminate or stop, examine returned statu
WTERMSIG	**wait**(2V)	wait for process to terminate or stop, examine returned statu

NAME

accept – accept a connection on a socket

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int accept(s, addr, addrlen)
int s;
struct sockaddr *addr;
int *addrlen;

DESCRIPTION

The argument *s* is a socket that has been created with **socket**(2), bound to an address with **bind**(2), and is listening for connections after a **listen**(2). **accept( )** extracts the first connection on the queue of pending connections, creates a new socket with the same properties of *s* and allocates a new file descriptor for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, **accept( )** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept( )** returns an error as described below. The accepted socket is used to read and write data to and from the socket which connected to this one; it is not used to accept more connections. The original socket *s* remains open for accepting further connections.

The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the domain in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with **SOCK_STREAM**.

It is possible to **select**(2) a socket for the purposes of doing an **accept( )** by selecting it for read.

RETURN VALUES

accept( ) returns a non-negative descriptor for the accepted socket on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EBADF	The descriptor is invalid.
EFAULT	The *addr* parameter is not in a writable part of the user address space.
ENOTSOCK	The descriptor references a file, not a socket.
EOPNOTSUPP	The referenced socket is not of type **SOCK_STREAM**.
EWOULDBLOCK	The socket is marked non-blocking and no connections are present to be accepted.

SEE ALSO

bind(2), **connect**(2), **listen**(2), **select**(2), **socket**(2)

NAME

access – determine accessibility of file

SYNOPSIS

#include <unistd.h>

int access(path, mode)
char *path;
int mode;

DESCRIPTION

path points to a path name naming a file. **access( )** checks the named file for accessibility according to *mode*, which is an inclusive or of the following bits:

R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute or search permission

The following value may also be supplied for *mode*:

F_OK	test whether the directories leading to the file can be searched and the file exists.

The real user ID and the supplementary group IDs (including the real group ID) are used in verifying permission, so this call is useful to set-UID programs.

Notice that only access bits are checked. A directory may be indicated as writable by **access( )**, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but **execve( )** will fail unless it is in proper format.

RETURN VALUES

access( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
	The file access permissions do not permit the requested access to the file named by *path*.
EFAULT	*path* points outside the process's allocated address space.
EINVAL	An invalid value was specified for *mode*.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file named by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EROFS	The file named by *path* is on a read-only file system and write access was requested.

SYSTEM V ERRORS

In addtion to the above, the following may also occur:

ENOENT	*path* points to an empty string.

SEE ALSO

chmod(2V), **stat**(2V)

NAME

acct – turn accounting on or off

SYNOPSIS

int acct (path)
char *path;

DESCRIPTION

acct() is used to enable or disable the process accounting. If process accounting is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an **exit()** call or a signal; see **exit (2V)** and **sigvec**(2). The effective user ID of the calling process must be super-user to use this call.

path points to a path name naming the accounting file. The accounting file format is given in **acct**(5).

The accounting routine is enabled if *path* is not a NULL pointer and no errors occur during the system call. It is disabled if *path* is a NULL pointer and no errors occur during the system call.

If accounting is already turned on, and a successful **acct()** call is made with a non-NULL *path*, all subsequent accounting records will be written to the new accounting file.

SYSTEM V DESCRIPTION

If accounting is already turned on, it is an error to call **acct()** with a non-NULL path.

RETURN VALUES

acct() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
	The file referred to by *path* is not a regular file.
EFAULT	*path* points outside the process's allocated address space.
EINVAL	Support for accounting was not configured into the system.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating the path name.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The named file does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EPERM	The caller is not the super-user.
EROFS	The named file resides on a read-only file system.

SYSTEM V ERRORS

EBUSY	*path* is non-NULL, and accounting is already turned on.
ENOENT	*path* points to an empty string.

SEE ALSO

exit(2V), **sigvec**(2), **acct**(5), **sa**(8)

BUGS

No accounting records are produced for programs running when a crash occurs. In particular non-terminating programs are never accounted for.

NOTES

Accounting is automatically disabled when free space on the file system the accounting file resides on drops below 2 percent; it is enabled when free space rises above 4 percent.

NAME

adjtime – correct the time to allow synchronization of the system clock

SYNOPSIS

```
#include <sys/time.h>

int adjtime(delta, olddelta)
struct timeval *delta;
struct timeval *olddelta;
```

DESCRIPTION

adjtime( ) adjusts the system's notion of the current time, as returned by **gettimeofday**(2), advancing or retarding it by the amount of time specified in the **struct timeval** (defined in **<sys/time.h>**) pointed to by *delta*.

The adjustment is effected by speeding up (if that amount of time is positive) or slowing down (if that amount of time is negative) the system's clock by some small percentage, generally a fraction of one percent. Thus, the time is always a monotonically increasing function. A time correction from an earlier call to **adjtime( )** may not be finished when **adjtime( )** is called again. If *olddelta* is not a NULL pointer, then the structure it points to will contain, upon return, the number of microseconds still to be corrected from the earlier call. If *olddelta* is a NULL pointer, the corresponding information will not be returned.

This call may be used in time servers that synchronize the clocks of computers in a local area network. Such time servers would slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time.

Only the super-user may adjust the time of day.

The adjustment value will be silently rounded to the resolution of the system clock.

RETURN

A 0 return value indicates that the call succeeded. A –1 return value indicates an error occurred, and in this case an error code is stored into the global variable **errno**.

ERRORS

EFAULT	*delta* or *olddelta* points outside the process's allocated address space.
	olddelta points to a region of the process' allocated address space that is not writable.
EPERM	The process's effective user ID is not that of the super-user.

SEE ALSO

date(1V), **gettimeofday**(2)

NAME

audit – write a record to the audit log

SYNOPSIS

#include <sys/label.h>
#include <sys/audit.h>

int audit (record)
audit_record_t *record;

DESCRIPTION

The **audit()** system call is used to write a record to the system audit log file. The data pointed to by *record* is written to the audit log file. The data should be a well-formed audit record as described by **audit.log**(5). The kernel sets the time stamp value in the record and performs a minimal check on the data before writing it to the audit log file.

Only the super-user may successfully execute this call.

RETURN VALUES

audit() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EFAULT *record* points outside the process's allocated address space.

EINVAL The length specified in the audit record is too short, or more than **MAXAUDITDATA**.

EPERM The process's effective user ID is not super-user.

SEE ALSO

auditsvc(2), **getauid**(2), **setuseraudit**(2), **audit_args**(3), **audit.log**(5), **auditd**(8)

NAME

auditon – manipulate auditing

SYNOPSIS

#include <sys/label.h>
#include <sys/audit.h>

int auditon (condition)
int condition;

DESCRIPTION

The **auditon()** system call sets system auditing to the requested *condition* if and only if the current state of auditing allows that transition. Legitimate values for *condition* are:

AUC_UNSET	on/off has not been decided yet
AUC_AUDITING	auditing is to be done
AUC_NOAUDIT	auditing is not to be done

The permitted transitions are:

- Any condition may be changed back to itself.
- AUC_UNSET may be changed to AUC_AUDITING or AUC_NOAUDIT.
- AUC_AUDITING may be changed to AUC_NOAUDIT.
- AUC_NOAUDIT may be changed to AUC_AUDITING.

Once changed, it is not possible to get back to AUC_UNSET.

Only the super-user may successfully execute this call.

RETURN VALUES

auditon() returns the old audit condition value on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EINVAL	The *condition* specified is outside the range of valid values.
	The current condition precludes the requested change.
EPERM	Neither of the process's effective or real user ID is super-user.

SEE ALSO

audit(2), **setuseraudit**(2)

NAME

auditsvc – write audit records to specified file descriptor

SYNOPSIS

int auditsvc(fd, limit)
int fd;
int limit;

DESCRIPTION

The **auditsvc()** system call specifies the audit log file to the kernel. The kernel writes audit records to this file until an exceptional condition occurs and then the call returns. The parameter *fd* is a file descriptor that identifies the audit file. Programs should open this file for writing before calling **auditsvc()**. The parameter *limit* specifies a value between 0 and 100, instructing **auditsvc()** to return when the percentage of free disk space on the audit filesystem drops below this limit. Thus, the invoking program can take action to avoid running out of disk space. The **auditsvc()** system call does not return until one of the following conditions occurs:

- The process receives a signal that is not blocked or ignored.
- An error is encountered writing to the audit log file.
- The minimum free space (as specified by *limit*), has been reached.

Only processes with a real or effective user ID of super-user may execute this call successfully.

RETURN VALUES

auditsvc() returns only on an error.

ERRORS

EAGAIN	The descriptor referred to a *stream*, was marked for System V-style non-blocking I/O, and no data could be written immediately.
EBADF	*fd* is not a valid descriptor open for writing.
EBUSY	A second process attempted to perform this call.
	A second process attempted to perform this call.
EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.
	Audit filesystem space is below the specified limit.
EFBIG	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.
EINTR	The call is forced to terminate prematurely due to the arrival of a signal whose **SV_INTERRUPT** bit in **sv_flags** is set (see **sigvec**(2)). **signal**(3V), in the System V compatibility library, sets this bit for any signal it catches.
EINVAL	Auditing is disabled (see **auditon**(2)).
	fd does not refer to a file of an appropriate type. Regular files are always appropriate.
EIO	An I/O error occurred while reading from or writing to the file system.
ENOSPC	There is no free space remaining on the file system containing the file.
ENXIO	A hangup occurred on the *stream* being written to.
EPERM	The process's effective or real user ID is not super-user.
EWOULDBLOCK	The file was marked for 4.2BSD-style non-blocking I/O, and no data could be written immediately.

SEE ALSO

audit(2), **auditon**(2), **sigvec**(2), **signal**(3V), **audit.log**(5), **auditd**(8)

NAME

bind – bind a name to a socket

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int bind(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;

DESCRIPTION

bind() assigns a name to an unnamed socket. When a socket is created with **socket**(2) it exists in a name space (address family) but has no name assigned. **bind()** requests that the name pointed to by *name* be assigned to the socket.

RETURN VALUES

bind() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES	The requested address is protected, and the current user has inadequate permission to access it.
EADDRINUSE	The specified address is already in use.
EADDRNOTAVAIL	The specified address is not available from the local machine.
EBADF	*s* is not a valid descriptor.
EFAULT	The *name* parameter is not in a valid part of the user address space.
EINVAL	*namelen* is not the size of a valid address for the specified address family.
	The socket is already bound to an address.
ENOTSOCK	*s* is a descriptor for a file, not a socket.

The following errors are specific to binding names in the UNIX domain:

EACCES	Search permission is denied for a component of the path prefix of the path name in *name*.
EIO	An I/O error occurred while making the directory entry or allocating the inode.
EISDIR	A null path name was specified.
ELOOP	Too many symbolic links were encountered in translating the path name in *name*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix of the path name in *name* does not exist.
ENOTDIR	A component of the path prefix of the path name in *name* is not a directory.
EROFS	The inode would reside on a read-only file system.

SEE ALSO

connect(2), **getsockname**(2), **listen**(2), **socket**(2), **unlink**(2V)

NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using **unlink**(2V),

The rules used in name binding vary between communication domains. Consult the manual entries in section 4 for detailed information.

NAME

brk, sbrk – change data segment size

SYNOPSIS

```
#include <sys/types.h>

int brk(addr)
caddr_t addr;

caddr_t sbrk(incr)
int incr;
```

DESCRIPTION

brk() sets the system's idea of the lowest data segment location not used by the program (called the *break*) to *addr* (rounded up to the next multiple of the system's page size).

In the alternate function **sbrk()**, *incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution using **execve()** the break is set at the highest location defined by the program and data storage areas.

The **getrlimit**(2) system call may be used to determine the maximum permissible size of the *data* segment; it will not be possible to set the break beyond the **rlim_max** value returned from a call to **getrlimit()**, that is to say, "**etext + rlim.rlim_max.**" (See **end**(3) for the definition of **etext()**.)

RETURN VALUES

brk() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

sbrk() returns the old break value on success. On failure, it returns **(caddr_t)** −1 and sets **errno** to indicate the error.

ERRORS

brk() and **sbrk()** will fail and no additional memory will be allocated if one of the following occurs:

ENOMEM The data segment size limit, as set by **setrlimit()** (see **getrlimit**(2)), would be exceeded.

The maximum possible size of a data segment (compiled into the system) would be exceeded.

Insufficient space exists in the swap area to support the expansion.

Out of address space; the new break value would extend into an area of the address space defined by some previously established mapping (see **mmap**(2)).

SEE ALSO

execve(2V), **mmap**(2), **getrlimit**(2), **malloc**(3V), **end**(3)

WARNINGS

Programs combining the **brk()** and **sbrk()** system calls and **malloc()** will not work. Many library routines use **malloc()** internally, so use **brk()** and **sbrk()** only when you know that **malloc()** definitely will not be used by any library routine.

BUGS

Setting the break may fail due to a temporary lack of swap space. It is not possible to distinguish this from a failure caused by exceeding the maximum size of the data segment without consulting **getrlimit()**.

NAME
chdir – change current working directory

SYNOPSIS
int chdir(path)
char *path;

int fchdir(fd)
int fd;

DESCRIPTION
chdir( ) and **fchdir( )** make the directory specified by *path* or *fd* the current working directory. Subsequent references to pathnames not starting with '/' are relative to the new current working directory.

In order for a directory to become the current directory, a process must have execute (search) access to the directory.

RETURN VALUES
chdir( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS
EACCES	Search permission is denied for a component of the pathname.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The named directory does not exist.
ENOTDIR	A component of the pathname is not a directory.

SYSTEM V ERRORS
In addition to the above, the following may also occur:

ENOENT	*path* points to an empty string.

WARNINGS
fchdir( ) is provided as a performance enhancement and is guaranteed to fail under certain conditions. In particular, if auditing is active the call will never succeed, and EINVAL will be returned. Applications which use this system call must be coded to detect this failure and switch to using **chdir( )** from that point on.

NAME

chmod, fchmod – change mode of file

SYNOPSIS

```
#include <sys/stat.h>

int chmod(path, mode)
char *path;
mode_t mode;

int fchmod(fd, mode)
int fd, mode;
```

DESCRIPTION

chmod( ) sets the mode of the file referred to by *path* or the descriptor *fd* according to *mode*. *mode* is the inclusive OR of the file mode bits (see **stat**(2V) for a description of these bits).

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user and the process attempts to set the set group ID bit on a file owned by a group which is not in its supplementary group IDs, the **S_ISGID** bit (set group ID on execution) is cleared.

If the **S_ISVTX** (sticky) bit is set on a directory, an unprivileged user may not delete or rename files of other users in that directory.

If a user other than the super-user writes to a file, the set user ID and set group ID bits are turned off. This makes the system somewhat more secure by protecting set-user-ID (set-group-ID) files from remaining set-user-ID (set-group-ID) if they are modified, at the expense of a degree of compatibility.

RETURN VALUES

chmod( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

chmod( ) will fail and the file mode will be unchanged if:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*path* points outside the process's allocated address space.
EINVAL	*fd* refers to a socket, not to a file.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EPERM	The effective user ID does not match the owner of the file and the effective user ID is not the super-user.
EROFS	The file referred to by *path* resides on a read-only file system.

fchmod( ) will fail if:

EBADF	The descriptor is not valid.

EIO	An I/O error occurred while reading from or writing to the file system.
EPERM	The effective user ID does not match the owner of the file and the effective user ID is not the super-user.
EROFS	The file referred to by *fd* resides on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to a null pathname.

SEE ALSO

chown(2V), **open**(2V), **stat**(2V), **sticky**(8)

BUGS

S_ISVTX, the "sticky bit", is a misnomer, and is overloaded to mean different things for different file types.

NAME

chown, fchown – change owner and group of a file

SYNOPSIS

```
int chown(path, owner, group)
char *path;
int owner;
int group;

int fchown(fd, owner, group)
int fd;
int owner;
int group;
```

SYSTEM V SYNOPSIS

```
#include <sys/types.h>

int chown(path, owner, group)
char *path;
uid_t owner;
gid_t group;
```

DESCRIPTION

The file that is named by *path* or referenced by *fd* has its *owner* and *group* changed as specified. Only the super-user may change the owner of the file, because if users were able to give files away, they could defeat the file-space accounting procedures (see NOTES). The owner of the file may change the group to a group of which he is a member. The super-user may change the group arbitrarily.

fchown() is particularly useful when used in conjunction with the file locking primitives (see **flock**(2)).

If *owner* or *group* is specified as –1, the corresponding ID of the file is not changed.

If a process whose effective user ID is not super-user successfully changes the group ID of a file, the set-user-ID and set-group-ID bits of the file mode, **S_ISUID** and **S_ISGID** respectively (see **stat**(2V)), will be cleared.

If the final component of *path* is a symbolic link, the ownership and group of the symbolic link is changed, not the ownership and group of the file or directory to which it points.

RETURN VALUES

chown() and **fchown()** return:

0	on success.
–1	on failure and set **errno** to indicate the error.

ERRORS

chown() will fail and the file will be unchanged if:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*path* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.

EPERM — The user ID specified by *owner* is not the current owner ID of the file.

The group ID specified by *group* is not the current group ID of the file and is not in the process' supplementary group IDs, and the effective user ID is not the super-user.

EROFS — The file referred to by *path* resides on a read-only file system.

fchown( ) will fail if:

EBADF — *fd* does not refer to a valid descriptor.

EINVAL — *fd* refers to a socket, not a file.

EIO — An I/O error occurred while reading from or writing to the file system.

EPERM — The user ID specified by *owner* is not the current owner ID of the file.

The group ID specified by *group* is not the current group ID of the file and is not in the supplementary group IDs, and the effective user ID is not the super-user.

EROFS — The file referred to by *fd* resides on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT — *path* points to an empty string.

SEE ALSO

chmod(2V), **flock**(2)

NOTES

For **chown**( ) to behave as described above, {_POSIX_CHOWN_RESTRICTED} must be in effect (see **pathconf**(2V)). {_POSIX_CHOWN_RESTRICTED} is always in effect on SunOS systems, but for portability, applications should call **pathconf**( ) to determine whether {_POSIX_CHOWN_RESTRICTED} is in effect for *path*.

If {_POSIX_CHOWN_RESTRICTED} is in effect for the file system on which the file referred to by *path* or *fd* resides, only the super-user may change the owner of the file. Otherwise, processes with effective user ID equal to the file owner or super-user may change the owner of the file.

NAME

chroot – change root directory

SYNOPSIS

int chroot(dirname)
char *dirname;

int fchroot(fd)
int fd;

DESCRIPTION

chroot() and **fchroot()** cause a directory to become the root directory, the starting point for path names beginning with '/'. The current working directory is unaffected by this call. This root directory setting is inherited across **execve**(2V) and by all children of this process created with **fork (2V)** calls.

In order for a directory to become the root directory a process must have execute (search) access to the directory and either the effective user ID of the process must be super-user or the target directory must be the system root or a loop-back mount of the system root (see **lofs**(4S)). **fchroot()** is further restricted in that while it is always possible to change to the system root using this call, it is not guaranteed to succeed in any other case, even should *fd* be in all respects valid.

The *dirname* argument to **chroot()** points to a path name of a directory. The *fd* argument to **fchroot()** is the open file descriptor of the directory which is to become the root.

The **..** entry in the root directory is interpreted to mean the root directory itself. Thus, **..** cannot be used to access files outside the subtree rooted at the root directory. Instead, **fchroot()** can be used to set the root back to a directory which was opened before the root directory was changed.

WARNINGS

The only use of **fchroot()** that is appropriate is to change back to the system root. While it may succeed in some other cases, it is guaranteed to fail if auditing is enabled. Super-user processes are not exempt from this limitation.

RETURN VALUES

chroot() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

chroot() will fail and the root directory will be unchanged if one or more of the following are true:

EACCES	Search permission is denied for a component of the path prefix of *dirname*.
	Search permission is denied for the directory referred to by *dirname*.
EBADF	The descriptor is not valid.
EFAULT	*dirname* points outside the process's allocated address space.
EINVAL	**fchroot()** attempted to change to a directory which is not the system root and external circumstances, such as auditing, do not allow this.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *dirname*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The directory referred to by *dirname* does not exist.

ENOTDIR A component of the path prefix of *dirname* is not a directory.

The file referred to by *dirname* is not a directory.

EPERM The effective user ID is not super-user.

SEE ALSO

chdir(2V), **execve**(2V), **fork**(2V), **lofs**(4S)

NAME

close – delete a descriptor

SYNOPSIS

int close (fd)
int fd;

DESCRIPTION

close() deletes a descriptor from the per-process object reference table. If *fd* is the last reference to the underlying object, then the object will be deactivated. For example, on the last close of a file the current *seek* pointer associated with the file is lost. On the last close of a socket (see **socket**(2)), associated naming information and queued data are discarded. On the last close of a file holding an advisory lock applied by **flock**(2), the lock is released. (Record locks applied to the file by **lockf**(3), however, are released on *any* call to **close()** regardless of whether *fd* is the last reference to the underlying object.)

close() does not unmap any mapped pages of the object referred to by *fd* (see **mmap()**, **munmap**(2)).

A close of all of a process's descriptors is automatic on **exit()**, but since there is a limit on the number of active descriptors per process, **close()** is necessary for programs that deal with many descriptors.

When a process forks (see **fork**(2v)), all descriptors for the new child process reference the same objects as they did in the parent before the fork. If a new process is then to be run using **execve**(2V), the process would normally inherit these descriptors. Most of the descriptors can be rearranged with **dup**(2V) or deleted with **close()** before the **execve()** is attempted, but if some of these descriptors will still be needed if the **execve()** fails, it is necessary to arrange for them to be closed if the **execve()** succeeds. The **fcntl**(2V) operation **F_SETFD** can be used to arrange that a descriptor will be closed after a successful **execve()**, or to restore the default behavior, which is to not close the descriptor.

If a STREAMS (see **intro**(2)) file is closed, and the calling process had previously registered to receive a **SIGPOLL** signal (see **sigvec**(2)) for events associated with that file (see **I_SETSIG** in **streamio**(4)), the calling process will be unregistered for events associated with the file. The last **close()** for a stream causes that stream to be dismantled. If the descriptor is not marked for no-delay mode and there have been no signals posted for the stream, **close()** waits up to 15 seconds, for each module and driver, for any output to drain before dismantling the stream. If the descriptor is marked for no-delay mode or if there are any pending signals, **close()** does not wait for output to drain, and dismantles the stream immediately.

RETURN VALUES

close() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EBADF *fd* is not an active descriptor.

EINTR A signal was caught before the close completed.

SEE ALSO

accept(2), **dup**(2V), **execve**(2V), **fcntl**(2V), **flock**(2), **intro**(2), **open**(2V), **pipe**(2V), **sigvec**(2), **socket**(2), **socketpair**(2), **streamio**(4)

NAME

connect – initiate a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

DESCRIPTION

The parameter *s* is a socket. If it is of type **SOCK_DGRAM**, then this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If it is of type **SOCK_STREAM**, then this call attempts to make a connection to another socket. The other socket is specified by *name* which is an address in the communications space of the socket. Each communications space interprets the *name* parameter in its own way. Generally, stream sockets may successfully **connect( )** only once; datagram sockets may use **connect( )** multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

RETURN VALUES

connect( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

The call fails if:

EADDRINUSE	The address is already in use.
EADDRNOTAVAIL	The specified address is not available on the remote machine.
EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
EALREADY	The socket is non-blocking and a previous connection attempt has not yet been completed.
EBADF	*s* is not a valid descriptor.
ECONNREFUSED	The attempt to connect was forcefully rejected. The calling program should **close**(2V) the socket descriptor, and issue another **socket**(2) call to obtain a new descriptor before attempting another **connect**(2) call.
EFAULT	The *name* parameter specifies an area outside the process address space.
EINPROGRESS	The socket is non-blocking and the connection cannot be completed immediately. It is possible to **select**(2) for completion by selecting the socket for writing.
EINTR	The connection attempt was interrupted before any data arrived by the delivery of a signal.
EINVAL	*namelen* is not the size of a valid address for the specified address family.
EISCONN	The socket is already connected.
ENETUNREACH	The network is not reachable from this host.
ENOTSOCK	*s* is a descriptor for a file, not a socket.
ETIMEDOUT	Connection establishment timed out without establishing a connection.

The following errors are specific to connecting names in the UNIX domain. These errors may not apply in future versions of the UNIX IPC domain.

EACCES — Search permission is denied for a component of the path prefix of the path name in *name*.

ELOOP — Too many symbolic links were encountered in translating the path name in *name*.

EIO — An I/O error occurred while reading from or writing to the file system.

ENAMETOOLONG — The length of the path argument exceeds {PATH_MAX}.

A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).

ENOENT — A component of the path prefix of the path name in *name* does not exist.

The socket referred to by the path name in *name* does not exist.

ENOTDIR — A component of the path prefix of the path name in *name* is not a directory.

ENOTSOCK — The file referred to by *name* is not a socket.

EPROTOTYPE — The file referred to by *name* is a socket of a type other than the type of *s* (e.g., *s* is a **SOCK_DGRAM** socket, while *name* refers to a **SOCK_STREAM** socket).

SEE ALSO

accept(2), **close**(2V), **connect**(2), **getsockname**(2), **select**(2), **socket**(2)

NAME

creat – create a new file

SYNOPSIS

int creat(path, mode)
char *path;
int mode;

SYSTEM V SYNOPSIS

#include <sys/stat.h>

int creat(path, mode)
char *path;
mode_t mode;

DESCRIPTION

This interface is made obsolete by **open**(2V), since,

creat(path, mode);

is equivalent to

open(path, O_WRONLY | O_CREAT | O_TRUNC, mode);

creat() creates a new ordinary file or prepares to rewrite an existing file named by the pathname pointed to by *path*. If the file did not exist, it is given the mode *mode*, as modified by the process's mode mask (see **umask**(2V)). See **stat**(2V) for the construction of *mode*.

If the file exists, its mode and owner remain unchanged, but it is truncated to 0 length. Otherwise, the file's owner ID is set to the effective user ID of the process, and upon successful completion, **creat()** marks for update the **st_atime**, **st_ctime**, and **st_mtime** fields of the file (see **stat**(2V)) and the **st_ctime** and **st_mtime** fields of the parent directory.

The file's group ID is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2V)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

The low-order 12 bits of the file mode are set to the value of *mode*, modified as follows:

- All bits set in the process's file mode creation mask are cleared. See **umask**(2V).
- The "save text image after execution" (sticky) bit of the mode is cleared. See **chmod**(2V).
- The "set group ID on execution" bit of the mode is cleared if the effective user ID of the process is not super-user and the process is not a member of the group of the created file.

Upon successful completion, the file descriptor is returned and the file is open for writing, even if the access permissions of the file mode do not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across **execve**(2V) system calls. See **fcntl**(2V).

If the file did not previously exist, upon successful completion, **creat()** marks for update the **st_ctime** and **st_mtime** fields of the file and the **st_ctime** and **st_mtime** fields of the parent directory.

RETURN VALUES

creat() returns a non-negative descriptor that only permits writing on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix.
	The file referred to by *path* does not exist and the directory in which it is to be created is not writable.
	The file referred to by *path* exists, but it is unwritable.

EDQUOT	The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
	The user's quota of inodes on the file system on which the file is being created has been exhausted.
EFAULT	*path* points outside the process's allocated address space.
EINTR	The **creat( )** operation was interrupted by a signal.
EIO	An I/O error occurred while making the directory entry or allocating the inode.
EISDIR	The file referred to by *path* is a directory.
ELOOP	Too many symbolic links were encountered in translating the pathname pointed to by *path*.
EMFILE	There are already too many files open.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENFILE	The system file table is full.
ENOENT	A component of the path prefix does not exist.
ENOSPC	The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.
	There are no free inodes on the file system on which the file is being created.
ENOTDIR	A component of the path prefix is not a directory.
ENXIO	The file is a character special or block special file, and the associated device does not exist.
EOPNOTSUPP	The file was a socket (not currently implemented).
EROFS	The file referred to by *path* resides, or would reside, on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to an empty string.

SEE ALSO

close(2V), **chmod**(2V), **execve**(2V), **fcntl**(2V), **flock**(2), **mount**(2V), **open**(2V), **write**(2V), **umask**(2V)

NOTES

The *mode* given is arbitrary; it need not allow writing. This feature has been used in the past by programs to construct a simple exclusive locking mechanism. It is replaced by the **O_EXCL** open mode, or **flock**(2) facility.

NAME

dup, dup2 – duplicate a descriptor

SYNOPSIS

int dup(fd)
int fd;

int dup2(fd1, fd2)
int fd1, fd2;

DESCRIPTION

dup() duplicates an existing object descriptor. The argument *fd* is a small non-negative integer index in the per-process descriptor table. The value must be less than the size of the table, which is returned by **getdtablesize**(2). The new descriptor returned by the call is the lowest numbered descriptor that is not currently in use by the process.

With **dup2()**, *fd2* specifies the desired value of the new descriptor. If descriptor *fd2* is already in use, it is first deallocated as if it were closed by **close**(2V).

The new descriptor has the following in common with the original:

- It refers to the same object that the old descriptor referred to.
- It uses the same seek pointer as the old descriptor. (that is, both file descriptors share one seek pointer).
- It has the same access mode (read, write or read/write) as the old descriptor.

Thus if *fd2* and *fd1* are duplicate references to an open file, **read**(2V), **write**(2V), and **lseek**(2V) calls all move a single seek pointer into the file, and append mode, non-blocking I/O and asynchronous I/O options are shared between the references. If a separate seek pointer into the file is desired, a different object reference to the file must be obtained by issuing an additional **open**(2V) call. The close-on-exec flag on the new file descriptor is unset.

The new file descriptor is set to remain open across **exec** system calls (see **fcntl**(2V).

RETURN VALUES

dup() and **dup2()** return a new descriptor on success. On failure, they return −1 and set **errno** to indicate the error.

ERRORS

EBADF	*fd1* or *fd2* is not a valid active descriptor.
EMFILE	Too many descriptors are active.

SEE ALSO

accept(2), **close**(2V), **fcntl**(2V), **getdtablesize**(2), **lseek**(2V), **open**(2V), **pipe**(2V), **read**(2V), **socket**(2), **socketpair**(2), **write**(2V)

NAME

execve – execute a file

SYNOPSIS

int execve(path, argv, envp)
char *path, *argv[ ], *envp[ ];

DESCRIPTION

execve( ) transforms the calling process into a new process. The new process is constructed from an ordinary file, whose name is pointed to by *path*, called the *new process file*. This file is either an executable object file, or a file of data for an interpreter. An executable object file consists of an identifying header, followed by pages of data representing the initial program (text) and initialized data pages. Additional pages may be specified by the header to be initialized with zero data. See **a.out**(5).

An interpreter file begins with a line of the form '**#!** *interpreter* [*arg*]'. Only the first thirty-two characters of this line are significant. When *path* refers to an interpreter file, **execve( )** invokes the specified *interpreter*. If the optional *arg* is specified, it becomes the first argument to the *interpreter*, and the pathname to which *path* points becomes the second argument. Otherwise, the pathname to which *path* points becomes the first argument. The original arguments are shifted over to become the subsequent arguments. The zeroth argument, normally the pathname to which *path* points, is left unchanged.

There can be no return from a successful **execve( )** because the calling process image is lost. This is the mechanism whereby different process images become active.

The argument *argv* is a pointer to a null-terminated array of character pointers to null-terminated character strings. These strings constitute the argument list to be made available to the new process. By convention, at least one argument must be present in this array, and the first element of this array should be the name of the executed program (that is, the last component of *path*).

The argument *envp* is also a pointer to a null-terminated array of character pointers to null-terminated strings. These strings pass information to the new process which are not directly arguments to the command (see **environ**(5V)).

The number of bytes available for the new process's combined argument and environment lists (including null terminators, pointers and alignment bytes) is {ARG_MAX} (see **sysconf**(2V)). On SunOS systems, {ARG_MAX} is currently one megabyte.

Descriptors open in the calling process remain open in the new process, except for those for which the close-on-exec flag is set (see **close**(2V) and **fcntl**(2V)). Descriptors which remain open are unaffected by **execve( )**.

Signals set to the default action (SIG_DFL) in the calling process image are set to the default action in the new process image. Signals set to be ignored (SIG_IGN) by the calling process image are ignored by the new process image. Signals set to be caught by the calling process image are reset to the default action in the new process image. Signals set to be blocked in the calling process image remain blocked in the new process image, regardless of changes to the signal action. The signal stack is reset to be undefined (see **sigvec**(2) for more information).

Each process has a *real* user ID and group ID and an *effective* user ID and group ID. The *real* ID identifies the person using the system; the *effective* ID determines their access privileges. **execve( )** changes the effective user or group ID to the owner or group of the executed file if the file has the "set-user-ID" or "set-group-ID" modes. The *real* UID and GID are not affected. The effective user ID and effective group ID of the new process image are saved as the saved set-user-ID and saved set-group-ID respectively, for use by **setuid**(3V).

execve( ) sets the SEXECED flag for the new process image (see **setpgid**(2V)).

The shared memory segments attached to the calling process will not be attached to the new process (see **shmop**(2)).

Profiling is disabled for the new process; see **profil**(2).

Upon successful completion, **execve**( ) marks for update the **st_atime** field of the file. **execve**( ) also marks **st_atime** for update if it fails, but is able find the process image file.

If **execve**( ) succeeds, the process image file is considered to have been opened (see **open**(2V)). The corresponding close (see **close**(2V)) is considered to occur after the open, but before process termination or successful completion of a subsequent call to **execve**( ).

The new process also inherits the following attributes from the calling process:

attribute	*see*
process ID	**getpid**(2)
parent process ID	**getpid**(2)
process group ID	**getpgrp**(2V), **setpgid**(2V)
session membership	**setsid**(2)
real user ID	**getuid**(2)
real group ID	**getgid**(2)
supplementary group IDs	**Intro**(2)
time left until an alarm	**alarm**(3C)
supplementary group IDs	**getgroups**(2)
semadj values	**semop**(2)
working directory	**chdir**(2)
root directory	**chroot**(2)
controlling terminal	**termio**(4)
trace flag	**ptrace**(2), request 0
resource usages	**getrusage**(2)
interval timers	**getitimer**(2)
resource limits	**getrlimit**(2)
file mode mask	**umask**(2)
process signal mask	**sigvec**(2), **sigprocmask**(2V), **sigsetmask**(2)
pending signals	**sigpending**(2)
tms_utime, **tms_stime**, **tms_cutime**, **tms_cstime**	**times**(3C)

When the executed program begins, it is called as follows:

```
main(argc, argv, envp)
int argc;
char *argv[ ], *envp[ ];
```

where *argc* is the number of elements in *argv* (the "arg count", not counting the NULL terminating pointer) and *argv* points to the array of character pointers to the arguments themselves.

envp is a pointer to an array of strings that constitute the *environment* of the process. A pointer to this array is also stored in the global variable **environ**. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a NULL pointer. The shell **sh**(1) passes an environment entry for each global shell variable defined when the program is called. See **environ**(5V) for some conventionally used names.

Note: Passing values for *argc*, *argv*, and *envp* to **main**( ) is optional.

RETURN VALUES

execve( ) returns to the calling process only on failure. It returns −1 and sets **errno** to indicate the error.

ERRORS

E2BIG — The total number of bytes in the new process file's argument and environment lists exceeds {ARG_MAX} (see **sysconf**(2V)).

EACCES — Search permission is denied for a component of the new process file's path prefix.

	The new process file is not an regular file.
	Execute permission is denied for the new process file.
EFAULT	The new process file is not as long as indicated by the size values in its header.
	path, *argv*, or *envp* points to an illegal address.
EIO	An I/O error occurred while reading from the file system.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ELOOP	Too many symbolic links were encountered in translating *path*.
ENOENT	One or more components of the path prefix of the new process file does not exist.
	The new process file does not exist.
ENOEXEC	The new process file has the appropriate access permission, but has an invalid magic number in its header.
ENOMEM	The new process file requires more virtual memory than is allowed by the imposed maximum (**getrlimit**(2)).
ENOTDIR	A component of the path prefix of the new process file is not a directory.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to a null pathname.

SEE ALSO

sh(1), **chdir**(2V), **chroot**(2), **close**(2V), **exit**(2V), **fcntl**(2V), **fork**(2V), **getgroups**(2V), **getitimer**(2), **getpid**(2V), **getrlimit**(2), **getrusage**(2), **profil**(2), **ptrace**(2), **semop**(2), **getpgrp**(2V), **shmop**(2), **sigvec**(2), **execl**(3V), **setuid**(3V), **termio**(4), **a.out**(5), **environ**(5V)

WARNINGS

If a program is **setuid**() to a non-super-user, but is executed when the real user ID is super-user, then the program has some of the powers of a super-user as well.

NAME

_exit – terminate a process

SYNOPSIS

void _exit(status)
int status;

DESCRIPTION

_exit( ) terminates a process with the following consequences:

All of the descriptors open in the calling process are closed. This may entail delays, for example, waiting for output to drain; a process in this state may not be killed, as it is already dying.

If the parent process of the calling process is executing a **wait( )** or **waitpid( )**, or is interested in the **SIGCHLD** signal, then it is notified of the calling process's termination and the low-order eight bits of *status* are made available to it (see **wait**(2V)).

If the parent process of the calling process is not executing a **wait( )** or **waitpid( )**, *status* is saved for return to the parent process whenever the parent process executes an appropriate subsequent **wait( )** or **waitpid( )**.

The parent process ID of all of the calling process's existing child processes are also set to 1. This means that the initialization process (see **intro**(2)) inherits each of these processes as well. Any stopped children are restarted with a hangup signal (**SIGHUP**).

If the process is a controlling process, **SIGHUP** is sent to each process in the foreground process group of the controlling terminal belonging to the calling process, and the controlling terminal associated with the session is disassociated from the session, allowing it to be acquired by a new controlling process (see **setsid**(2V)).

If **_exit( )** causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then **SIGHUP** followed by **SIGCONT** is sent to each process in the newly-orphaned process group (see **setpgid**(2V)).

Each attached shared memory segment is detached and the value of **shm_nattach** in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a *semadj* value (see **semop**(2)), that *semadj* value is added to the *semval* of the specified semaphore.

If process accounting is enabled (see **acct**(2V)), an accounting record is written to the accounting file.

Most C programs will call the library routine **exit**(3) which performs cleanup actions in the standard I/O library before calling **_exit( )**.

RETURN VALUES

_exit( ) never returns.

SEE ALSO

intro(2), **acct**(2V), **fork**(2V), **semop**(2), **wait**(2V), **exit**(3)

NAME

fcntl – file control

SYNOPSIS

#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl(fd, cmd, arg)
int fd, cmd, arg;

DESCRIPTION

fcntl() performs a variety of functions on open descriptors. The argument *fd* is an open descriptor used by *cmd* as follows:

F_DUPFD Returns a new descriptor, which has the smallest value greater than or equal to *arg*. It refers to the same object as the original descriptor, and has the same access mode (read, write or read/write). The new descriptor shares descriptor status flags with *fd*, and if the object was a file, the same file pointer. It is also associated with a **FD_CLOEXEC** (close-on-exec) flag set to remain open across **execve**(2V) system calls.

F_GETFD Get the **FD_CLOEXEC** (close-on-exec) flag associated with *fd*. If the low-order bit is **0**, the file remains open after executing **execve()**, otherwise it is closed.

F_SETFD Set the **FD_CLOEXEC** (close-on-exec) flag associated with *fd* to the low order bit of *arg* (**0** or **1** as above).

Note: this is a per-process and per-descriptor flag. Setting or clearing it for a particular descriptor does not affect the flag on descriptors copied from it by **dup**(2V) or **F_DUPFD**, nor does it affect the flag on other processes of that descriptor.

F_GETFL Get descriptor status flags (see **fcntl**(5) for definitions).

F_SETFL Set descriptor status flags (see **fcntl**(5) for definitions). The following flags are the only ones whose values may change: **O_APPEND**, **O_SYNC**, and **O_NDELAY**, and the **FASYNC**, **FNDELAY**, and **FNBIO** flags defined in **<fcntl.h>**.

O_NDELAY and **FNDELAY** are identical.

Descriptor status flag values set by **F_SETFL** affects descriptors copied using **dup**(2V), **F_DUPFD** or other processes.

Setting or clearing the **FNDELAY** flag on a descriptor causes an **FIONBIO ioctl**(2) request to be performed on the object referred to by that descriptor. Setting or clearing non-blocking mode, and setting or clearing the **FASYNC** flag on a descriptor causes an **FIOASYNC ioctl**(2) request to be performed on the object referred to by that descriptor, setting or clearing asynchronous mode. Thus, all descriptors referring to the object are affected.

F_GETLK Get a description of the first lock which would block the lock specified by the **flock** structure pointed to by *arg* (see the definition of **struct flock** below). If a lock exists, The **flock** structure is overwritten with that lock's description. Otherwise, the structure is passed back with the lock type set to **F_UNLOCK** and is otherwise unchanged.

F_SETLK Set or clear a file segment lock according to the **flock** structure pointed to by *arg*. **F_SETLK** is used to set shared (**F_RDLCK**) or exclusive (**F_WRLCK**) locks, or to remove those locks (**F_UNLCK**). If the specified lock cannot be applied, **fcntl()** fails and returns immediately.

F_SETLKW This *cmd* is the same as **F_SETLK** except that if a shared or exclusive lock is blocked by other locks, the process waits until the requested lock can be applied. If a signal that is set to be caught (see **signal**(3V)) is received while **fcntl**( ) is waiting for a region, the call to **fcntl**( ) is interrupted. Upon return from the process's signal handler, **fcntl**( ) fails and returns, and the requested lock is not applied.

F_GETOWN Get the process ID or process group currently receiving **SIGIO** and **SIGURG** signals; process groups are returned as negative values.

F_SETOWN Set the process or process group to receive **SIGIO** and **SIGURG** signals. Process groups are specified by supplying *arg* as negative, otherwise *arg* is interpreted as a process ID.

F_RSETLK
F_RSETLKW
F_RGETLK Are used by the network lock daemon, **lockd**(8C), to communicate with the NFS server kernel to handle locks on the NFS files.

Record locking is done with either *shared* (**F_RDLCK**), or *exclusive* (**F_WRLCK**) locks. More than one process may hold a shared lock on a particular file segment, but if one process holds an exclusive lock on the segment, no other process may hold any lock on the segment until the exclusive lock is removed.

In order to claim a shared lock, a descriptor must be opened with read access. Descriptors for exclusive locks must be opened with write access.

A shared lock may be changed to an exclusive lock, and vice versa, simply by specifying the appropriate lock type with a **F_SETLK** or **F_SETLKW** *cmd*. Before the previous lock is released and the new lock applied, any other processes already in line must gain and release their locks.

If *cmd* is **F_SETLKW** and the requested lock cannot be claimed immediately (for instance, when another process holds an exclusive lock that overlaps the current request) the calling process is blocked until the lock may be acquired. These blocks may be interrupted by signals. Care should be taken to avoid deadlocks caused by multiple processes all blocking the same records.

A shared or exclusive lock is either *advisory* or *mandatory* depending on the mode bits of the file containing the locked segment. The lock is mandatory if the set-GID bit (**S_ISGID**) is set and the group execute bit (**S_IXGRP**) is clear (see **stat**(2V) for information about mode bits). Otherwise, the lock is advisory.

If a process holds a mandatory shared lock on a segment of a file, other processes may read from the segment, but write operations block until all locks are removed. If a process holds a mandatory exclusive lock on a segment of a file, both read and write operations block until the lock is removed (see **WARNINGS**).

An advisory lock does not affect read and write access to the locked segment. Advisory locks may be used by cooperating processes checking for locks using **F_GETLCK** and voluntarily observing the indicated read and write restrictions.

The record to be locked or unlocked is described by the **flock** structure defined in **<fcntl.h>** as follows:

```
struct flock {
        short l_type;    /* F_RDLCK, F_WRLCK, or F_UNLCK */
        short l_whence;  /* flag to choose starting offset */
        long  l_start;   /* relative offset, in bytes */
        long  l_len;     /* length, in bytes; 0 means lock to EOF */
        pid_t l_pid;     /* returned with F_GETLK */
};
```

The **flock** structure describes the type (**l_type**), starting offset (**l_whence**), relative offset (**l_start**), and size (**l_len**) of the file segment to be affected. **l_whence** is set to **SEEK_SET**, **SEEK_CUR**, or **SEEK_END** (see **lseek**(2V)) to indicate that the relative offset is to be measured from the start of the file, current position, or EOF, respectively. The process id field (**l_pid**) is only used with the **F_GETLK** *cmd* to return the description of a lock held by another process. Note: do not confuse **struct flock** with the function **flock**(2). They are unrelated.

Locks may start or extend beyond the current EOF, but may not be negative relative to the beginning of the file. Setting **l_len** to zero (0) extends the lock to EOF. If **l_whence** is set to **SEEK_SET** and **l_start** and **l_len** are set to zero (0), the entire file is locked. Changing or unlocking the subset of a locked segment leaves the smaller segments at either end locked. Locking a segment already locked by the calling process causes the old lock type to be removed and the new lock type to take affect. All locks associated with a file for a given process are removed when the file is closed or the process terminates. Locks are not inherited by the child process in a **fork**(2V) system call.

fcntl( ) record locks are implemented in the kernel for local locks, and throughout the network by the network lock daemon (**lockd**(8C)) for remote locks on NFS files. If the file server crashes and has to be rebooted, the lock daemon attempts to recover all locks that were associated with that server. If a lock cannot be reclaimed, the process that held the lock is issued a **SIGLOST** signal.

In order to maintain consistency in the network case, data must not be cached on client machines. For this reason, file buffering for an NFS file is turned off when the first lock is attempted on the file. Buffering remains off as long as the file is open. Programs that do I/O buffering in the user address space, however, may have inconsistent results. The standard I/O package, for instance, is a common source of unexpected buffering.

SYSTEM V DESCRIPTION

O_NDELAY and **FNBIO** are identical.

RETURN VALUES

On success, the value returned by **fcntl**( ) depends on *cmd* as follows:

F_DUPFD	A new descriptor.
F_GETFD	Value of flag (only the low-order bit is defined).
F_GETFL	Value of flags.
F_GETOWN	Value of descriptor owner.
other	Value other than −1.

On failure, **fcntl**( ) returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES — *cmd* is **F_SETLK**, the lock type (**l_type**) is **F_RDLCK** (shared lock), and the file segment to be locked is already under an exclusive lock held by another process. This error is also returned if the lock type is **F_WRLCK** (exclusive lock) and the file segment is already locked with a shared or exclusive lock.

Note: In future, **fcntl**( ) may generate EAGAIN under these conditions, so applications testing for EACCES should also test for EAGAIN.

EBADF — *fd* is not a valid open descriptor.

cmd is **F_SETLK** or **F_SETLKW** and the process does not have the appropriate read or write permissions on the file.

EDEADLK — *cmd* is **F_SETLKW**, the lock is blocked by one from another process, and putting the calling-process to sleep would cause a deadlock.

EFAULT — *cmd* is **F_GETLK, F_SETLK**, or **F_SETLKW** and *arg* points to an invalid address.

EINTR — *cmd* is **F_SETLKW** and a signal interrupted the process while it was waiting for the lock to be granted.

EINVAL — *cmd* is **F_DUPFD** and *arg* is negative or greater than the maximum allowable number (see **getdtablesize**(2)).

cmd is **F_GETLK, F_SETLK**, or **F_SETLKW** and *arg* points to invalid data.

EMFILE *cmd* is **F_DUPFD** and the maximum number of open descriptors has been reached.

ENOLCK *cmd* is **F_SETLK** or **F_SETLKW** and there are no more file lock entries available.

SEE ALSO

close(2V), **execve**(2V), **flock**(2), **fork**(2V), **getdtablesize**(2), **ioctl**(2), **open**(2V), **sigvec**(2), **lockf**(3), **fcntl**(5), **lockd**(8C)

WARNINGS

Mandatory record locks are dangerous. If a runaway or otherwise out-of-control process should hold a mandatory lock on a file critical to the system and fail to release that lock, the entire system could hang or crash. For this reason, mandatory record locks may be removed in a future SunOS release. Use advisory record locking whenever possible.

NOTES

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access. Files can be accessed without advisory files, but inconsistencies may result.

read(2V) and **write**(2V) system calls on files are affected by mandatory file and record locks (see **chmod**(2V)).

BUGS

File locks obtained by **fcntl()** do not interact with **flock()** locks. They do, however, work correctly with the exclusive locks claimed by **lockf**(3).

F_GETLK returns **F_UNLCK** if the requesting process holds the specified lock. Thus, there is no way for a process to determine if it is still holding a specific lock after catching a **SIGLOST** signal.

In a network environment, the value of **l_pid** returned by **F_GETLK** is next to useless.

NAME

flock – apply or remove an advisory lock on an open file

SYNOPSIS

```
#include <sys/file.h>

#define LOCK_SH    1    /* shared lock */
#define LOCK_EX    2    /* exclusive lock */
#define LOCK_NB    4    /* don't block when locking */
#define LOCK_UN    8    /* unlock */

int flock(fd, operation)
int fd, operation;
```

DESCRIPTION

flock() applies or removes an *advisory* lock on the file associated with the file descriptor *fd*. A lock is applied by specifying an *operation* parameter that is the inclusive OR of **LOCK_SH** or **LOCK_EX** and, possibly, **LOCK_NB**. To unlock an existing lock, the *operation* should be **LOCK_UN**.

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (that is, processes may still access files without using advisory locks, possibly resulting in inconsistencies).

The locking mechanism allows two types of locks: *shared* locks and *exclusive* locks. More than one process may hold a shared lock for a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on a file.

A shared lock may be *upgraded* to an exclusive lock, and vice versa, simply by specifying the appropriate lock type; the previous lock will be released and the new lock applied (possibly after other processes have gained and released the lock).

Requesting a lock on an object that is already locked normally causes the caller to block until the lock may be acquired. If **LOCK_NB** is included in *operation*, then this will not happen; instead the call will fail and the error EWOULDBLOCK will be returned.

NOTES

Locks are on files, not file descriptors. That is, file descriptors duplicated through **dup**(2V) or **fork**(2V)) do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock.

Processes blocked awaiting a lock may be awakened by signals.

RETURN VALUES

flock() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EBADF	The argument **fd** is an invalid descriptor.
EOPNOTSUPP	The argument **fd** refers to an object other than a file.
EWOULDBLOCK	The file is locked and the **LOCK_NB** option was specified.

SEE ALSO

close(2V), **dup**(2V), **execve**(2V), **fcntl**(2V), **fork**(2V), **open**(2V), **lockf**(3), **lockd**(8C)

BUGS

Locks obtained through the **flock()** mechanism are known only within the system on which they were placed. Thus, multiple clients may successfully acquire exclusive locks on the same remote file. If this behavior is not explicitly desired, the **fcntl**(2V) or **lockf**(3) system calls should be used instead; these make use of the services of the **network lock manager** (see **lockd**(8C)).

NAME

fork – create a new process

SYNOPSIS

int fork()

SYSTEM V SYNOPSIS

pid_t fork()

DESCRIPTION

fork() creates a new process. The new process (child process) is an exact copy of the calling process except for the following:

- The child process has a unique process ID. The child process ID also does not match any active process group ID.
- The child process has a different parent process ID (the process ID of the parent process).
- The child process has its own copy of the parent's descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an **lseek**(2V)) on a descriptor in the child process can affect a subsequent **read**(2V) or **write**(2V) by the parent. This descriptor copying is also used by the shell to establish standard input and output for newly created processes as well as to set up pipes.
- The child process has its own copy of the parent's open directory streams (see **directory**(3V)). Each open directory stream in the child process shares directory stream positioning with the corresponding directory stream of the parent.
- All *semadj* values are cleared; see **semop**(2).
- The child processes resource utilizations are set to 0; see **getrlimit**(2). The **it_value** and **it_interval** values for the **ITIMER_REAL** timer are reset to 0; see **getitimer**(2).
- The child process's values of **tms_utime**(), **tms_stime**(), **tms_cutime**(), and **tms_cstime**() (see **times**(3V)) are set to zero.
- File locks (see **fcntl**(2V)) previously set by the parent are not inherited by the child.
- Pending alarms (see **alarm**(3V)) are cleared for the child process.
- The set of signals pending for the child process is cleared (see **sigvec**(2)).

RETURN VALUES

On success, **fork()** returns 0 to the child process and returns the process ID of the child process to the parent process. On failure, **fork()** returns −1 to the parent process, sets **errno** to indicate the error, and no child process is created.

ERRORS

fork() will fail and no child process will be created if one or more of the following are true:

EAGAIN	The system-imposed limit on the total number of processes under execution would be exceeded. This limit is determined when the system is generated.
	The system-imposed limit on the total number of processes under execution by a single user would be exceeded. This limit is determined when the system is generated.
ENOMEM	There is insufficient swap space for the new process.

SEE ALSO

execve(2V), **getitimer**(2), **getrlimit**(2), **lseek**(2V), **read**(2V), **semop**(2), **wait**(2V), **write**(2V)

NAME

fsync – synchronize a file's in-core state with that on disk

SYNOPSIS

int fsync(fd)

int fd;

DESCRIPTION

fsync() moves all modified data and attributes of *fd* to a permanent storage device: all in-core modified copies of buffers for the associated file have been written to a disk when the call returns. Note: this is different than **sync**(2) which schedules disk I/O for all files (as though an **fsync()** had been done on all files) but returns before the I/O completes.

fsync() should be used by programs which require a file to be in a known state; for example, a program which contains a simple transaction facility might use it to ensure that all modifications to a file or files caused by a transaction were recorded on disk.

RETURN VALUES

fsync() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EBADF	*fd* is not a valid descriptor.
EINVAL	*fd* refers to a socket, not a file.
EIO	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

cron(8), **sync**(2)

NAME

getauid, setauid – get and set user audit identity

SYNOPSIS

int getauid()

int setauid(auid)
int auid;

DESCRIPTION

The **getauid()** system call returns the audit user ID for the current process. This value is initially set at login time and inherited by all child processes. This value does not change when the real/effective user IDs change, so it can be used to identify the logged-in user, even when running a setuid program. The audit user ID governs audit decisions for a process.

The **setauid()** system call sets the audit user ID for the current process. Only the super-user may successfully execute these calls.

RETURN VALUES

getauid() returns the audit user ID of the current process on success. On failure, it returns −1 and sets **errno** to indicate the error.

setauid() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL The parameter *auid* is not a valid UID.

EPERM The process's effective user ID is not super-user.

SEE ALSO

getuid(2V), **setuseraudit**(2), **audit**(8)

NAME

getdents – gets directory entries in a filesystem independent format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dirent.h>

int getdents(fd, buf, nbytes)
int fd;
char *buf;
int nbytes;
```

DESCRIPTION

getdents( ) attempts to put directory entries from the directory referenced by the file descriptor *fd* into the buffer pointed to by *buf*, in a filesystem independent format. Up to *nbytes* bytes of data will be transferred.

The data in the buffer is a series of **dirent** structures each containing the following entries:

```
off_t       d_off;
u_long      d_fileno;
u_short     d_reclen;
u_short     d_namlen;
char        d_name[MAXNAMLEN + 1];  /* see below */
```

The **d_off** entry contains a value which is interpretable only by the filesystem that generated it. It may be supplied as an offset to **lseek**(2V)) to find the entry following the current one in a directory. The **d_fileno** entry is a number which is unique for each distinct file in the filesystem. Files that are linked by hard links (see **link**(2V)) have the same **d_fileno**. The **d_reclen** entry is the length, in bytes, of the directory record. The **d_name** entry contains a null terminated file name. The **d_namlen** entry specifies the length of the file name. Thus the actual size of **d_name** may vary from 1 to **MAXNAMLEN+1**.

The structures are not necessarily tightly packed. The **d_reclen** entry may be used as an offset from the beginning of a *dirent* structure to the next structure, if any.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with *fd* is set to point to the directory entry following the last one returned. The pointer is not necessarily incremented by the number of bytes returned by **getdents( )**. If the value returned is zero, the end of the directory has been reached. The current position pointer may be set and retrieved by **lseek**(2V). It is not safe to set the current position pointer to any value other than a value previously returned by **lseek**(2V), or the value of a **d_off** entry in a **dirent** structure returned by **getdents**( ), or zero.

RETURN VALUES

getdents( ) returns the number of bytes actually transferred on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EBADF	*fd* is not a valid file descriptor open for reading.
EFAULT	*buf* points outside the allocated address space.
EINTR	A read from a slow device was interrupted before any data arrived by the delivery of a signal.
EINVAL	*nbytes* is not large enough for one directory entry.
ENOTDIR	The file referenced by *fd* is not a directory.
EIO	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

link(2V), **lseek**(2V), **open**(2V), **directory**(3V)

NOTES

It is strongly recommended, for portability reasons, that programs that deal with directory entries use the **directory**(3V) interface rather than directly calling **getdents**( ).

NAME

getdirentries – gets directory entries in a filesystem independent format

SYNOPSIS

```
int getdirentries(fd, buf, nbytes, basep)
int fd;
char *buf;
int nbytes;
long *basep;
```

DESCRIPTION

This system call is now obsolete. It is superseded by the **getdents**(2) system call, which returns directory entries in a new format specified in **<sys/dirent.h>**. The file, **<sys/dir.h>**, has also been modified to use the new directory entry format. Programs which currently call **getdirentries( )** should be modified to use the new system call and the new include file **dirent.h** or, preferably, to use the **directory**(3V) library routines. The **getdirentries( )** system call is retained in the current SunOS release only for purposes of backwards binary compatibility and will be removed in a future major release.

getdirentries( ) attempts to put directory entries from the directory referenced by the file descriptor **fd** into the buffer pointed to by *buf*, in a filesystem independent format. Up to *nbytes* bytes of data will be transferred. *nbytes* must be greater than or equal to the block size associated with the file, see **stat**(2V). Sizes less than this may cause errors on certain filesystems.

The data in the buffer is a series of structures each containing the following entries:

```
unsigned long  d_fileno;
unsigned short d_reclen;
unsigned short d_namlen;
char           d_name[MAXNAMELEN + 1]; /* see below */
```

The **d_fileno** entry is a number which is unique for each distinct file in the filesystem. Files that are linked by hard links (see **link**(2V)) have the same **d_fileno**. The **d_reclen** entry is the length, in bytes, of the directory record. The **d_name** entry contains a null terminated file name. The **d_namlen** entry specifies the length of the file name. Thus the actual size of **d_name** may vary from 2 to MAXNAMELEN+1.

The structures are not necessarily tightly packed. The **d_reclen** entry may be used as an offset from the beginning of a **direct** structure to the next structure, if any.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with **fd** is set to point to the next block of entries. The pointer is not necessarily incremented by the number of bytes returned by **getdirentries**( ). If the value returned is zero, the end of the directory has been reached. The current position pointer may be set and retrieved by **lseek**(2V). **getdirentries( )** writes the position of the block read into the location pointed to by *basep*. It is not safe to set the current position pointer to any value other than a value previously returned by **lseek**(2V) or a value previously returned in the location pointed to by *basep* or zero.

RETURN VALUES

getdirentries( ) returns the number of bytes actually transferred on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EBADF	*fd* is not a valid file descriptor open for reading.
EFAULT	Either *buf* or *basep* points outside the allocated address space.
EINTR	A read from a slow device was interrupted before any data arrived by the delivery of a signal.
EIO	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

getdents(2), **link**(2V), **lseek**(2V), **open**(2V), **stat**(2V), **directory**(3V)

NAME

getdomainname, setdomainname – get/set name of current domain

SYNOPSIS

int getdomainname(name, namelen)
char *name;
int namelen;

int setdomainname(name, namelen)
char *name;
int namelen;

DESCRIPTION

getdomainname() returns the name of the domain for the current processor, as previously set by **setdomainname.** The parameter *namelen* specifies the size of the array pointed to by *name*. The returned name is null-terminated unless insufficient space is provided.

setdomainname() sets the domain of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

The purpose of domains is to enable two distinct networks that may have host names in common to merge. Each network would be distinguished by having a different domain name. At the current time, only the Network Information Service (NIS) and **sendmail**(8) make use of domains.

RETURN VALUES

getdomainname() and **setdomainname()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

EFAULT The *name* parameter gave an invalid address.

In addition to the above, **setdomainname()** will fail if:

EPERM The caller was not the super-user.

NOTES

Domain names are limited to 64 characters.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

getdtablesize – get descriptor table size

SYNOPSIS

getdtablesize( )

DESCRIPTION

The call **getdtablesize( )** returns the current value of the soft limit component of the **RLIMIT_NOFILE** resource limit. This resource limit governs the maximum value allowable as the index of a newly created descriptor.

WARNINGS

getdtablesize is implemented as a system call only for binary compatibility with previous releases.

Because of possible intervening **getrlimit**(2) calls affecting **RLIMIT_NOFILE**, repeated calls to **getdtablesize( )** may return different values. Thus it is unwise to cache the return value in an effort to avoid system call overhead, unless it is known that such intervening calls do not occur.

SEE ALSO

close(2V), **dup**(2V), **getrlimit**(2), **open**(2V)

NAME

getgid, getegid – get group identity

SYNOPSIS

int getgid( )

int getegid( )

SYSTEM V SYNOPSIS

#include <sys/types.h>

gid_t getgid( )

gid_t getegid( )

DESCRIPTION

getgid( ) returns the real group ID of the current process. **getegid( )** returns the effective group ID of the current process.

The GID is specified at login time by the group field in the **/etc/passwd** database (see **passwd**(5)).

The effective GID is more transient, and determines additional access permission during execution of a set-GID process, and it is for such processes that **getegid( )** is most useful.

SEE ALSO

getuid(2V), **setregid**(2), **setuid**(3V)

NAME

getgroups, setgroups – get or set supplementary group IDs

SYNOPSIS

```
int getgroups(gidsetlen, gidset)
int gidsetlen;
int gidset[];

int setgroups(ngroups, gidset)
int ngroups;
int gidset[];
```

SYSTEM V SYNOPSIS

```
#include <sys/types.h>

int getgroups(gidsetlen, gidset)
int gidsetlen;
gid_t gidset[];

int setgroups(ngroups, gidset)
int ngroups;
gid_t gidset[];
```

DESCRIPTION

getgroups( ) gets the current supplementary group IDs of the user process and stores it in the array *gidset*. The parameter *gidsetlen* indicates the number of entries that may be placed in *gidset*. **getgroups**( ) returns the actual number of entries placed in the *gidset* array. No more than {NGROUPS_MAX} (see **sysconf**(2V)), will ever be returned. If *gidsetlen* is 0, **getgroups**( ) returns the number of groups without modifying the *gidset* array.

setgroups( ) sets the supplementary group IDs of the current user process according to the array *gidset*. The parameter *ngroups* indicates the number of entries in the array and must be no more than {NGROUPS_MAX} (see **sysconf**(2V)).

Only the super-user may set new groups.

RETURN VALUES

On success, **getgroups**( ) returns the number of entries placed in the array pointed to by *gidset*. On failure, it returns –1 and sets **errno** to indicate the error.

setgroups( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

Either call fails if:

EFAULT The address specified for *gidset* is outside the process address space.

getgroups( ) fails if:

EINVAL The argument *gidsetlen* is smaller than the number of groups in the group set.

setgroups( ) fails if:

EPERM The caller is not the super-user.

SEE ALSO

initgroups(3)

NAME

gethostid – get unique identifier of current host

SYNOPSIS

gethostid()

DESCRIPTION

gethostid( ) returns the 32-bit identifier for the current host, which should be unique across all hosts. On a Sun workstation, this number is taken from the CPU board's ID PROM.

SEE ALSO

hostid(1)

NAME

gethostname, sethostname – get/set name of current host

SYNOPSIS

```
int gethostname(name, namelen)
char *name;
int namelen;

int sethostname(name, namelen)
char *name;
int namelen;
```

DESCRIPTION

gethostname() returns the standard host name for the current processor, as previously set by **sethostname()**. The parameter *namelen* specifies the size of the array pointed to by *name*. The returned name is null-terminated unless insufficient space is provided.

sethostname() sets the name of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

RETURN VALUES

gethostname() and **sethostname()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

EFAULT The *name* or *namelen* parameter gave an invalid address.

In addition to the above, **sethostname()** may set **errno** to:

EPERM The caller was not the super-user.

SEE ALSO

gethostid(2)

NOTES

Host names are limited to **MAXHOSTNAMELEN** (from <**sys/param.h**>) characters, currently 64.

NAME

getitimer, setitimer – get/set value of interval timer

SYNOPSIS

#include <sys/time.h>

int getitimer (which, value)
int which;
struct itimerval *value;

int setitimer (which, value, ovalue)
int which;
struct itimerval *value, *ovalue;

DESCRIPTION

The system provides each process with three interval timers, defined in **<sys/time.h>**. The **getitimer( )** call stores the current value of the timer specified by **which** into the structure pointed to by *value*. The **setitimer( )** call sets the value of the timer specified by **which** to the value specified in the structure pointed to by *value*, and if *ovalue* is not a NULL pointer, stores the previous value of the timer in the structure pointed to by *ovalue*.

A timer value is defined by the **itimerval** structure, which includes the following members:

struct timevalit_interval;/* timer interval */
struct timevalit_value; /* current value */

If **it_value** is non-zero, it indicates the time to the next timer expiration. If **it_interval** is non-zero, it specifies a value to be used in reloading **it_value** when the timer expires. Setting **it_value** to zero disables a timer; however, **it_value** and **it_interval** must still be initialized. Setting **it_interval** to zero causes a timer to be disabled after its next expiration (assuming **it_value** is non-zero).

Time values smaller than the resolution of the system clock are rounded up to this resolution.

The three timers are:

ITIMER_REAL Decrements in real time. A **SIGALRM** signal is delivered when this timer expires.

ITIMER_VIRTUAL Decrements in process virtual time. It runs only when the process is executing. A **SIGVTALRM** signal is delivered when it expires.

ITIMER_PROF Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the **ITIMER_PROF** timer expires, the **SIGPROF** signal is delivered. Because this signal may interrupt in-progress system calls, programs using this timer must be prepared to restart interrupted system calls.

RETURN VALUES

getitimer( ) and **setitimer( )** return:

0 on success.

–1 on failure and set **errno** to indicate the error.

ERRORS

The possible errors are:

EFAULT The *value* or *ovalue* parameter specified a bad address.

EINVAL The *value* parameter specified a time that was too large to be handled.

SEE ALSO

sigvec(2), **gettimeofday**(2)

NOTES

Three macros for manipulating time values are defined in **<sys/time.h>**. **timerclear** sets a time value to zero, **timerisset** tests if a time value is non-zero, and **timercmp** compares two time values (beware that >= and <= do not work with this macro).

NAME

getmsg – get next message from a stream

SYNOPSIS

#include <stropts.h>

int getmsg(fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int *flags;

DESCRIPTION

getmsg() retrieves the contents of a message (see **intro**(2)) located at the **stream head** read queue from a STREAMS file, and places the contents into user specified buffer(s). The message must contain either a data part, a control part or both. The data and control parts of the message are placed into separate buffers, as described below. The semantics of each part is defined by the STREAMS module that generated the message.

fd specifies a file descriptor referencing an open **stream**. *ctlptr* and *dataptr* each point to a **strbuf** structure that contains the following members:

```
int maxlen;     /* maximum buffer length */
int len;        /* length of data */
char *buf;      /* ptr to buffer */
```

where **buf** points to a buffer in which the data or control information is to be placed, and **maxlen** indicates the maximum number of bytes this buffer can hold. On return, **len** contains the number of bytes of data or control information actually received, or is 0 if there is a zero-length control or data part, or is –1 if no data or control information is present in the message. *flags* may be set to the values 0 or **RS_HIPRI** and is used as described below.

ctlptr is used to hold the control part from the message and *dataptr* is used to hold the data part from the message. If *ctlptr* (or *dataptr*) is a NULL pointer or the **maxlen** field is –1, the control (or data) part of the message is not processed and is left on the **stream head** read queue and **len** is set to –1. If the **maxlen** field is set to 0 and there is a zero-length control (or data) part, that zero-length part is removed from the read queue and **len** is set to 0. If the *maxlen* field is set to 0 and there are more than zero bytes of control (or data) information, that information is left on the read queue and **len** is set to 0. If the *maxlen* field in *ctlptr* or *dataptr* is less than, respectively, the control or data part of the message, *maxlen* bytes are retrieved. In this case, the remainder of the message is left on the **stream head** read queue and a non-zero return value is provided, as described below under RETURN VALUES. If information is retrieved from a **priority** message, *flags* is set to **RS_HIPRI** on return.

By default, **getmsg()** processes the first priority or non-priority message available on the **stream head** read queue. However, a process may choose to retrieve only priority messages by setting *flags* to **RS_HIPRI**. In this case, **getmsg()** will only process the next message if it is a priority message.

If **O_NDELAY** has not been set, **getmsg()** blocks until a message, of the type(s) specified by *flags* (priority or either), is available on the **stream head** read queue. If **O_NDELAY** has been set and a message of the specified type(s) is not present on the read queue, **getmsg()** fails and sets **errno** to EAGAIN.

If a hangup occurs on the **stream** from which messages are to be retrieved, **getmsg()** will continue to operate normally, as described above, until the **stream head** read queue is empty. Thereafter, it will return 0 in the **len** fields of *ctlptr* and *dataptr*.

RETURN VALUES

getmsg( ) returns a non-negative value on success:

0	A full message was read successfully.
MORECTL	More control information is waiting for retrieval. Subsequent **getmsg( )** calls will retrieve the rest of the message.
MOREDATA	More data are waiting for retrieval. Subsequent **getmsg( )** calls will retrieve the rest of the message.
MORECTL | MOREDATA	Both types of information remain.

On failure, **getmsg( )** returns −1 and sets **errno** to indicate the error.

ERRORS

EAGAIN	The **O_NDELAY** flag is set, and no messages are available.
EBADF	*fd* is not a valid file descriptor open for reading.
EBADMSG	The queued message to be read is not valid for **getmsg( )**.
EFAULT	*ctlptr*, *dataptr*, or *flags* points to a location outside the allocated address space.
EINTR	A signal was caught during the **getmsg( )** system call.
EINVAL	An illegal value was specified in *flags*.
	The **stream** referenced by *fd* is linked under a multiplexor.
ENOSTR	A **stream** is not associated with *fd*.

A **getmsg( )** can also fail if a STREAMS error message had been received at the **stream head** before the call to **getmsg( )**. The error returned is the value contained in the STREAMS error message.

SEE ALSO

intro(2), **poll**(2), **putmsg**(2), **read**(2V), **write**(2V)

NAME

getpagesize – get system page size

SYNOPSIS

int getpagesize( )

DESCRIPTION

getpagesize( ) returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls.

The page size is a *system* page size and may not be the same as the underlying hardware page size.

SEE ALSO

pagesize(1), **brk**(2)

NAME

getpeername – get name of connected peer

SYNOPSIS

int getpeername(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;

DESCRIPTION

getpeername( ) returns the name of the peer connected to socket *s*. The **int** pointed to by the *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

DIAGNOSTICS

A 0 is returned if the call succeeds, –1 if it fails.

ERRORS

EBADF	The argument *s* is not a valid descriptor.
EFAULT	The *name* parameter points to memory not in a valid part of the process address space.
ENOBUFS	Insufficient resources were available in the system to perform the operation.
ENOTCONN	The socket is not connected.
ENOTSOCK	The argument *s* is a file, not a socket.

SEE ALSO

accept(2), **bind**(2), **getsockname**(2), **socket**(2)

NAME

getpgrp, setpgrp – return or set the process group of a process

SYNOPSIS

int getpgrp(pid)
int pid;

int setpgrp(pid, pgrp)
int pgrp;
int pid;

SYSTEM V SYNOPSIS

int getpgrp( )

int setpgrp( )

DESCRIPTION

getpgrp( ) returns the process group of the process indicated by *pid*. If *pid* is zero, then the call applies to the calling process.

Process groups are used for distribution of signals, and by terminals to arbitrate requests for their input. Processes that have the same process group as the terminal run in the foreground and may read from the terminal, while others block with a signal when they attempt to read.

This call is thus used by programs such as **csh**(1) to create process groups in implementing job control. The **TIOCGPGRP** and **TIOCSPGRP** calls described in **termio**(4) are used to get/set the process group of the control terminal.

setpgrp( ) sets the process group of the specified process, (*pid*) to the process group specified by *pgrp*. If *pid* is zero, then the call applies to the current (calling) process. If *pgrp* is zero and *pid* refers to the calling process, **setpgrp( )** behaves identically to **setsid**(2V).

If the effective user ID of the calling process is not super-user, then the process to be affected must have the same effective user ID as that of the calling process or be a member of the same session as the calling process.

SYSTEM V DESCRIPTION

getpgrp( ) returns the process group of the calling process.

setpgrp( ) behaves identically to **setsid( )**.

RETURN VALUES

getpgrp( ) returns the process group of the indicated process on success. On failure, it returns −1 and sets **errno** to indicate the error.

setpgrp( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

SYSTEM V RETURN VALUES

getpgrp( ) returns the process group of the calling process on success.

ERRORS

setpgrp( ) fails, and the process group is not altered when one of the following occurs:

EACCES The value of *pid* matches the process ID of a child process of the calling process and the child process has successfully executed one of the **exec( )** functions.

EINVAL The value of *pgrp* is less than zero or is greater than MAXPID, the maximum process ID as defined in **<sys/param.h>**.

EPERM The process indicated by *pid* is a session leader.

The value of *pid* is valid but matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.

The value of *pgrp* does not match the process ID of the process indicated by *pid* and there is no process with a process group ID that matches the value of *pgrp* in the same session as the calling process.

The requested process has a different effective user ID from that of the calling process and is not a descendent of the calling process.

The calling process is already a process group leader

The process ID of the calling process equals the process group ID of a different process.

ESRCH The value of *pid* does not match the process ID of the calling process or of a child process of the calling process.

The requested process does not exist.

SEE ALSO

csh(1), **intro**(2), **execve**(2V), **fork**(2V), **getpid**(2V), **getuid**(2V), **kill**(2V), **setpgid**(2V), **signal**(3V), **termio**(4)

NAME

getpid, getppid – get process identification

SYNOPSIS

int getpid()

int getppid()

SYSTEM V SYNOPSIS

#include <sys/types.h>

pid_t getpid()

pid_t getppid()

DESCRIPTION

getpid() returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

getppid() returns the process ID of the parent of the current process.

SEE ALSO

gethostid(2)

NAME

getpriority, setpriority – get/set process nice value

SYNOPSIS

#include <sys/time.h>
#include <sys/resource.h>

int getpriority(which, who)
int which, who;

int setpriority(which, who, niceval)
int which, who, niceval;

DESCRIPTION

The nice value of a process, process group, or user, as indicated by *which* and *who* is obtained with the **getpriority( )** call and set with the **setpriority( )** call. Process nice values can range from –20 through 19. The default nice value is 0; lower nice values cause more favorable scheduling.

which is one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**, and *who* is interpreted relative to *which* (a process identifier for **PRIO_PROCESS**, process group identifier for **PRIO_PGRP**, and a user ID for **PRIO_USER**). A zero value of *who* denotes the current process, process group, or user.

The **getpriority( )** call returns the lowest numerical nice value of any of the specified processes. The **setpriority( )** call sets the nice values of all of the specified processes to the value specified by *niceval*. If *niceval* is less than –20, a value of –20 is used; if it is greater than 19, a value of 19 is used. Only the super-user may use negative nice values.

RETURN VALUES

Since **getpriority( )** can legitimately return the value –1, it is necessary to clear the external variable **errno** prior to the call, then check it afterward to determine if a –1 is an error or a legitimate value.

setpriority( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

getpriority( ) and **setpriority( )** may set **errno** to:

EINVAL *which* was not one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**.

ESRCH No process was located using the *which* and *who* values specified.

In addition to the errors indicated above, **setpriority( )** may fail with one of the following errors returned:

EACCES The call to **setpriority( )** would have changed a process' nice value to a value lower than its current value, and the effective user ID of the process executing the call was not that of the super-user.

EPERM A process was located, but neither its effective nor real user ID matched the effective user ID of the caller, and neither the effective nor the real user ID of the process executing **setpriority( )** was super-user.

SEE ALSO

nice(1), **ps**(1), **fork**(2V), **nice**(3v) **renice**(8)

BUGS

It is not possible for the process executing **setpriority( )** to lower any other process down to its current nice value, without requiring super-user privileges.

These system calls are misnamed. They get and set the nice value, not the kernel scheduling priority. **nice**(1) discusses the relationship between nice value and scheduling priority.

NAME

getrlimit, setrlimit – control maximum system resource consumption

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(resource, rlp)
int resource;
struct rlimit *rlp;

int setrlimit(resource, rlp)
int resource;
struct rlimit *rlp;
```

DESCRIPTION

Limits on the consumption of system resources by the current process and each process it creates may be obtained with the **getrlimit( )** call, and set with the **setrlimit( )** call.

The *resource* parameter is one of the following:

RLIMIT_CPU the maximum amount of cpu time (in seconds) to be used by each process.

RLIMIT_FSIZE the largest size, in bytes, of any single file that may be created.

RLIMIT_DATA the maximum size, in bytes, of the data segment for a process; this defines how far a program may extend its break with the **sbrk( )** (see **brk**(2)) system call.

RLIMIT_STACK the maximum size, in bytes, of the stack segment for a process; this defines how far a program's stack segment may be extended automatically by the system.

RLIMIT_CORE the largest size, in bytes, of a **core** file that may be created.

RLIMIT_RSS the maximum size, in bytes, to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size.

RLIMIT_NOFILE one more than the maximum value that the system may assign to a newly created descriptor. This limit constrains the number of descriptors that a process may create.

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded a process may receive a signal (for example, if the cpu time is exceeded), but it will be allowed to continue execution until it reaches the hard limit (or modifies its resource limit). The **rlimit** structure is used to specify the hard and soft limits on a resource,

```
struct rlimit {
        int     rlim_cur;       /* current (soft) limit */
        int     rlim_max;       /* hard limit */
};
```

Only the super-user may raise the maximum limits. Other users may only alter **rlim_cur** within the range from 0 to **rlim_max** or (irreversibly) lower **rlim_max**.

An "infinite" value for a limit is defined as **RLIM_INFINITY (0x7fffffff)**.

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; **limit** is thus a built-in command to **csh**(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a **brk()** or **sbrk()** call will fail if the data space limit is reached, or the process will be sent a **SIGSEGV** when the stack limit is reached which will kill the process unless **SIGSEGV** is handled on a separate signal stack (since the stack cannot be extended, there is no way to send a signal!).

A file I/O operation that would create a file that is too large generates a signal **SIGXFSZ**; this normally terminates the process, but may be caught. When the soft CPU time limit is exceeded, a signal **SIGXCPU** is sent to the offending process.

RETURN VALUES

getrlimit() and **setrlimit()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

EFAULT The address specified by *rlp* was invalid.

EINVAL An invalid *resource* was specified.

In addition to the above, **setrlimit()** may set **errno** to:

EINVAL The new **rlim_cur** exceeds the new **rlim_max**.

EPERM The limit specified would have raised the maximum limit value, and the caller was not the super-user.

SEE ALSO

csh(1), **sh**(1), **brk**(2), **getdtablesize**(2), **quotactl**(2)

BUGS

There should be **limit** and **unlimit** commands in **sh**(1) as well as in **csh**(1).

NAME

getrusage – get information about resource utilization

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getrusage(who, rusage)
int who;
struct rusage *rusage;
```

DESCRIPTION

getrusage( ) returns information about the resources utilized by the current process, or all its terminated child processes. The interpretation for some values reported, such as **ru_idrss**, are dependent on the clock tick interval. This interval is an implementation dependent value; for example, on Sun-3 sytems the clock tick interval is 1/50 of a second, while on Sun-4 systems the clock tick interval is 1/100 of a second.

The *who* parameter is one of **RUSAGE_SELF** or **RUSAGE_CHILDREN**. The buffer to which *rusage* points will be filled in with the following structure:

```
struct  rusage {
        struct timeval ru_utime;        /* user time used */
        struct timeval ru_stime;        /* system time used */
        int     ru_maxrss;              /* maximum resident set size */
        int     ru_ixrss;               /* currently 0 */
        int     ru_idrss;               /* integral resident set size */
        int     ru_isrss;               /* currently 0 */
        int     ru_minflt;              /* page faults not requiring physical I/O */
        int     ru_majflt;              /* page faults requiring physical I/O */
        int     ru_nswap;               /* swaps */
        int     ru_inblock;             /* block input operations */
        int     ru_oublock;             /* block output operations */
        int     ru_msgsnd;              /* messages sent */
        int     ru_msgrcv;              /* messages received */
        int     ru_nsignals;            /* signals received */
        int     ru_nvcsw;               /* voluntary context switches */
        int     ru_nivcsw;              /* involuntary context switches */
};
```

The fields are interpreted as follows:

ru_utime The total amount of time spent executing in user mode. Time is given in seconds and microseconds.

ru_stime The total amount of time spent executing in system mode. Time is given in seconds and microseconds.

ru_maxrss The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the **getpagesize**(2) system call). Also, see WARNINGS.

ru_ixrss Currently returns 0.

ru_idrss An "integral" value indicating the amount of memory in use by a process while the process is running. This value is the sum of the resident set sizes of the process running when a clock tick occurs. The value is given in pages times clock ticks. Note: it does not take sharing into account. Also, see WARNINGS.

ru_isrss Currently returns 0.

ru_minflt The number of page faults serviced which did not require any physical I/O activity. Also, see WARNINGS.

ru_majflt The number of page faults serviced which required physical I/O activity. This could include page ahead operations by the kernel. Also, see WARNINGS.

ru_nswap The number of times a process was swapped out of main memory.

ru_inblock The number of times the file system had to perform input in servicing a **read**(2V) request.

ru_oublock The number of times the file system had to perform output in servicing a **write**(2V) request.

ru_msgsnd The number of messages sent over sockets.

ru_msgrcv The number of messages received from sockets.

ru_nsignals The number of signals delivered.

ru_nvcsw The number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).

ru_nivcsw The number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

RETURN VALUES

getrusage( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EFAULT The address specified by the *rusage* argument is not in a valid portion of the process's address space.

EINVAL The *who* parameter is not a valid value.

SEE ALSO

gettimeofday(2), **read**(2V), **wait**(2V), **write**(2V)

WARNINGS

The numbers **ru_inblock** and **ru_oublock** account only for real I/O, and are approximate measures at best. Data supplied by the caching mechanism is charged only to the first process to read and the last process to write the data.

The way resident set size is calculated is an approximation, and could misrepresent the true resident set size.

Page faults can be generated from a variety of sources and for a variety of reasons. The customary cause for a page fault is a direct reference by the program to a page which is not in memory. Now, however, the kernel can generate page faults on behalf of the user, for example, servicing **read**(2V) and **write**(2V) system calls. Also, a page fault can be caused by an absent hardware translation to a page, even though the page is in physical memory.

In addition to hardware detected page faults, the kernel may cause pseudo page faults in order to perform some housekeeping. For example, the kernel may generate page faults, even if the pages exist in physical memory, in order to lock down pages involved in a raw I/O request.

By definition, *major* page faults require physical I/O, while *minor* page faults do not require physical I/O. For example, reclaiming the page from the free list would avoid I/O and generate a minor page fault. More commonly, minor page faults occur during process startup as references to pages which are already in memory. For example, if an address space faults on some "hot" executable or shared library, this results in a minor page fault for the address space. Also, any one doing a **read**(2V) or **write**(2V) to something that is in the page cache will get a minor page fault(s) as well.

BUGS

There is no way to obtain information about a child process which has not yet terminated.

NAME

getsockname – get socket name

SYNOPSIS

getsockname(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;

DESCRIPTION

getsockname( ) returns the current *name* for the specified socket. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes).

DIAGNOSTICS

A 0 is returned if the call succeeds, –1 if it fails.

ERRORS

The call succeeds unless:

EBADF	*s* is not a valid descriptor.
EFAULT	*name* points to memory not in a valid part of the process address space.
ENOBUFS	Insufficient resources were available in the system to perform the operation.
ENOTSOCK	*s* is a file, not a socket.

SEE ALSO

bind(2), **getpeername**(2), **socket**(2)

BUGS

Names bound to sockets in the UNIX domain are inaccessible; **getsockname( )** returns a zero length name.

NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

int setsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int optlen;
```

DESCRIPTION

getsockopt( ) and **setsockopt( )** manipulate *options* associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, *level* is specified as **SOL_SOCKET**. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, *level* should be set to the protocol number of TCP; see **getprotoent**(3N).

The parameters *optval* and *optlen* are used to access option values for **setsockopt( )**. For **getsockopt( )** they identify a buffer in which the value for the requested option(s) are to be returned. For **getsockopt( )**, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, *optval* may be supplied as 0.

optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file **<sys/socket.h>** contains definitions for "socket" level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section (4P).

Most socket-level options take an *int* parameter for *optval*. For **setsockopt( )**, the parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. **SO_LINGER** uses a **struct linger** parameter, defined in **<sys/socket.h>**, which specifies the desired state of the option and the linger interval (see below).

The following options are recognized at the socket level. Except as noted, each may be examined with **getsockopt( )** and set with **setsockopt( )**.

SO_DEBUG	toggle recording of debugging information
SO_REUSEADDR	toggle local address reuse
SO_KEEPALIVE	toggle keep connections alive
SO_DONTROUTE	toggle routing bypass for outgoing messages
SO_LINGER	linger on close if data present
SO_BROADCAST	toggle permission to transmit broadcast messages
SO_OOBINLINE	toggle reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a **bind**(2) call should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken. A process attempting to write to the socket receives a **SIGPIPE** signal and the write operation returns an error. By default, a process exits when it receives **SIGPIPE**. A read operation on the socket returns an error but does not generate **SIGPIPE**. If the process is waiting in **select**(2) when the connection is broken, **select**( ) returns true for any read or write events selected for the socket. **SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messags are queued on socket and a **close**(2V) is performed. If the socket promises reliable delivery of data and **SO_LINGER** is set, the system will block the process on the **close**( ) attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the **setsockopt**( ) call when **SO_LINGER** is requested). If **SO_LINGER** is disabled and a **close**( ) is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

The option **SO_BROADCAST** requests permission to send broadcast datagrams on the socket. Broadcast was a privileged operation in earlier versions of the system. With protocols that support out-of-band data, the **SO_OOBINLINE** option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with **recv**( ) or **read**( ) calls without the **MSG_OOB** flag. **SO_SNDBUF** and **SO_RCVBUF** are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values. Finally, **SO_TYPE** and **SO_ERROR** are options used only with **getsockopt**( ). **SO_TYPE** returns the type of the socket, such as **SOCK_STREAM**; it is useful for servers that inherit sockets on startup. **SO_ERROR** returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUES

getsockopt( ) and **setsockopt**( ) return:

0	on success.
–1	on failure and set **errno** to indicate the error.

ERRORS

EBADF	*s* is not a valid descriptor.
EFAULT	The address pointed to by *optval* is not in a valid part of the process address space.
ENOPROTOOPT	The option is unknown at the level indicated.
ENOTSOCK	*s* is a file, not a socket.

In addition to the above, **getsockopt**( ) may set **errno** to:

EFAULT	*optlen* is not in a valid part of the process address space.

SEE ALSO

ioctl(2), **socket**(2), **getprotoent**(3N)

BUGS

Several of the socket options should be handled at lower levels of the system.

NAME

gettimeofday, settimeofday – get or set the date and time

SYNOPSIS

```
#include <sys/time.h>

int gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;

int settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the **gettimeofday( )** call, and set with the **settimeofday( )** call. The current time is expressed in elapsed seconds and microseconds since 00:00 GMT, January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent; the time may be updated continuously, or in "ticks."

tp points to a **timeval** structure, which includes the following members:

```
long tv_sec;  /* seconds since Jan. 1, 1970 */
long tv_usec; /* and microseconds */
```

If *tp* is a NULL pointer, the current time information is not returned or set.

tzp points to a **timezone( )** structure, which includes the following members:

```
int tz_minuteswest; /* of Greenwich */
int tz_dsttime;     /* type of dst correction to apply */
```

The **timezone( )** structure indicates the local time zone (measured in minutes westward from Greenwich), and a flag that indicates the type of Daylight Saving Time correction to apply. Note: this flag does *not* indicate whether Daylight Saving Time is currently in effect.

Also note that the offset of the local time zone from GMT may change over time, as may the rules for Daylight Saving Time correction. The **localtime( )** routine (see **ctime**(3V)) obtains this information from a file rather than from **gettimeofday( )**. Programs should use **localtime( )** to convert dates and times; the **timezone( )** structure is filled in by **gettimeofday( )** for backward compatibility with existing programs.

The flag indicating the type of Daylight Saving Time correction should have one of the following values (as defined in **<sys/time.h>**):

0	**DST_NONE**: Daylight Savings Time not observed
1	**DST_USA**: United States DST
2	**DST_AUST**: Australian DST
3	**DST_WET**: Western European DST
4	**DST_MET**: Middle European DST
5	**DST_EET**: Eastern European DST
6	**DST_CAN**: Canadian DST
7	**DST_GB**: Great Britain and Eire DST
8	**DST_RUM**: Rumanian DST
9	**DST_TUR**: Turkish DST
10	**DST_AUSTALT**: Australian-style DST with shift in 1986

If *tzp* is a NULL pointer, the time zone information is not returned or set.

Only the super-user may set the time of day or the time zone.

RETURN VALUES

gettimeofday( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EFAULT An argument address referenced invalid memory.

EPERM A user other than the super-user attempted to set the time or time zone.

SEE ALSO

date(1V), **adjtime**(2), **ctime**(3V)

BUGS

Time is never correct enough to believe the microsecond values. There should a mechanism by which, at least, local clusters of systems might synchronize their clocks to millisecond granularity.

NAME

getuid, geteuid – get user identity

SYNOPSIS

int getuid( )

int geteuid( )

SYSTEM V SYNOPSIS

#include <sys/types.h>

uid_t getuid( )

uid_t geteuid( )

DESCRIPTION

getuid( ) returns the real user ID of the current process, **geteuid( )** the effective user ID.

The real user ID identifies the person who is logged in. The effective user ID gives the process different permissions during execution of "set-user-ID" mode processes, which use **getuid( )** to determine the real-user-id of the process that invoked them.

SEE ALSO

getgid(2V), **setreuid**(2)

NAME

ioctl – control device

SYNOPSIS

int ioctl(fd, request, arg)
int fd, request;
caddr_t arg;

DESCRIPTION

ioctl( ) performs a special function on the object referred to by the open descriptor *fd*. The set of functions that may be performed depends on the object that *fd* refers to. For example, many operating characteristics of character special files (for instance, terminals) may be controlled with **ioctl( )** requests. The writeups in section 4 discuss how **ioctl( )** applies to various objects.

The *request* codes for particular functions are specified in include files specific to objects or to families of objects; the writeups in section 4 indicate which include files specify which *request*s.

For most **ioctl( )** functions, *arg* is a pointer to data to be used by the function or to be filled in by the function. Other functions may ignore *arg* or may treat it directly as a data item; they may, for example, be passed an **int** value.

RETURN VALUES

ioctl( ) returns 0 on success for most requests. Some specialized requests may return non-zero values on success; see the description of the request in the man page for the object. On failure, **ioctl( )** returns −1 and sets **errno** to indicate the error.

ERRORS

EBADF	*fd* is not a valid descriptor.
EFAULT	*request* requires a data transfer to or from a buffer pointed to by *arg*, but some part of the buffer is outside the process's allocated space.
EINVAL	*request* or *arg* is not valid.
ENOTTY	The specified request does not apply to the kind of object to which the descriptor *fd* refers.

ioctl( ) will also fail if the object on which the function is being performed detects an error. In this case, an error code specific to the object and the function will be returned.

SEE ALSO

execve(2V), **fcntl**(2V), **filio**(4), **mtio**(4), **sockio**(4), **streamio**(4), **termio**(4)

NAME

kill – send a signal to a process or a group of processes

SYNOPSIS

```
#include <signal.h>

int kill(pid, sig)
int pid;
int sig;
```

SYSTEM V SYNOPSIS

```
#include <signal.h>

int kill(pid, sig)
pid_t pid;
int sig;
```

DESCRIPTION

kill() sends the signal *sig* to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. *sig* may be one of the signals specified in **sigvec**(2), or it may be 0, in which case error checking is performed but no signal is actually sent. This can be used to check the validity of *pid* or the existence of process *pid*.

The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is super-user. A single exception is the signal **SIGCONT**, which may always be sent to any member of the same session as the current process.

In the following discussion, "system processes" are processes, such as processes 0 and 2, that are not running a regular user program.

If *pid* is greater than zero, the signal is sent to the process whose process ID is equal to *pid*. *pid* may equal 1.

If *pid* is 0, the signal is sent to all processes, except system processes and process 1, whose process group ID is equal to the process group ID of the sender; this is a variant of **killpg**(2).

If *pid* is −1 and the effective user ID of the sender is not super-user, the signal is sent to all processes, except system processes, process 1, and the process sending the signal, whose real or saved set-user ID matches the real or effective ID of the sender.

If *pid* is −1 and the effective user ID of the sender is super-user, the signal is sent to all processes except system processes, process 1, and the process sending the signal.

If *pid* is negative but not −1, the signal is sent to all processes, except system processes, process 1, and the process sending the signal, whose process group ID is equal to the absolute value of *pid*; this is a variant of **killpg**(2).

Processes may send signals to themselves.

SYSTEM V DESCRIPTION

If a signal is sent to a group of processes (as with, if *pid* is 0 or negative), and if the process sending the signal is a member of that group, the signal is sent to that process as well.

The signal **SIGKILL** cannot be sent to process 1.

RETURN VALUES

kill() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

kill( ) will fail and no signal will be sent if any of the following occur:

EINVAL *sig* was not a valid signal number.

EPERM The effective user ID of the sending process was not super-user, and neither its real nor effective user ID matched the real or saved set-user ID of the receiving process.

ESRCH No process could be found corresponding to that specified by *pid*.

SYSTEM V ERRORS

kill( ) will also fail, and no signal will be sent, if the following occurs:

EINVAL *sig* is SIGKILL and *pid* is 1.

SEE ALSO

getpid(2V), **killpg**(2), **getpgrp**(2V), **sigvec**(2), **termio**(4)

NAME

killpg – send signal to a process group

SYNOPSIS

int killpg(pgrp, sig)
int pgrp, sig;

DESCRIPTION

killpg() sends the signal *sig* to the process group *pgrp*. See **sigvec**(2) for a list of signals.

The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is super-user. A single exception is the signal **SIGCONT**, which may always be sent to any descendant of the current process.

RETURN VALUES

killpg() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

killpg() will fail and no signal will be sent if any of the following occur:

EINVAL *sig* was not a valid signal number.

EPERM The effective user ID of the sending process was not super-user, and neither its real nor effective user ID matched the real or saved set-user ID of one or more of the target processes.

ESRCH No processes were found in the specified process group.

SEE ALSO

kill(2V), **getpgrp**(2V), **sigvec**(2)

NAME

link – make a hard link to a file

SYNOPSIS

int link(path1, path2)
char *path1, *path2;

DESCRIPTION

path1 points to a pathname naming an existing file. *path2* points to a pathname naming a new directory entry to be created. **link( )** atomically creates a new link for the existing file and increments the link count of the file by one. {LINK_MAX} (see **pathconf**(2V)) specifies the maximum allowed number of links to the file.

With hard links, both files must be on the same file system. Both the old and the new link share equal access and rights to the underlying object. The super-user may make multiple links to a directory. Unless the caller is the super-user, the file named by *path1* must not be a directory.

Upon successful completion, **link( )** marks for update the **st_ctime** field of the file. Also, the **st_ctime** and **st_mtime** fields of the directory that contains the new entry are marked for update.

RETURN VALUES

link( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

link( ) will fail and no link will be created if one or more of the following are true:

EACCES	Search permission is denied for a component of the path prefix pointed to by *path1* or *path2*.
	The requested link requires writing in a directory for which write permission is denied.
EDQUOT	The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
EEXIST	The link referred to by *path2* exists.
EFAULT	One of the path names specified is outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system to make the directory entry.
ELOOP	Too many symbolic links were encountered in translating the pathname pointed to by *path1* or *path2*.
EMLINK	The number of links to the file named by *path1* would exceed {LINK_MAX} (see **pathconf**(2V)).
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix pointed to by *path1* or *path2* does not exist.
	The file referred to by *path1* does not exist.
ENOSPC	The directory in which the entry for the new link is being placed cannot be extended because there is no space left on the file system containing the directory.
ENOTDIR	A component of the path prefix of *path1* or *path2* is not a directory.

EPERM	The file named by *path1* is a directory and the effective user ID is not super-user.
EROFS	The requested link requires writing in a directory on a read-only file system.
EXDEV	The link named by *path2* and the file named by *path1* are on different file systems.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path1* or *path2* points to an empty string.

SEE ALSO

symlink(2), **unlink**(2V)

NAME

listen – listen for connections on a socket

SYNOPSIS

int listen(s, backlog)
int s, backlog;

DESCRIPTION

To accept connections, a socket is first created with **socket**(2), a backlog for incoming connections is specified with **listen**( ) and then the connections are accepted with **accept**(2). The **listen**( ) call applies only to sockets of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client will receive an error with an indication of ECONNREFUSED.

RETURN VALUES

listen( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EBADF	*s* is not a valid descriptor.
ENOTSOCK	*s* is not a socket.
EOPNOTSUPP	The socket is not of a type that supports **listen**( ).

SEE ALSO

accept(2), **connect**(2), **socket**(2)

BUGS

The *backlog* is currently limited (silently) to 5.

NAME

lseek, tell – move read/write pointer

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(fd, offset, whence)
int fd;
off_t offset;
int whence;

long tell(fd)
int fd;
```

DESCRIPTION

lseek() sets the seek pointer associated with the open file or device referred to by the descriptor *fd* according to the value supplied for *whence*. *whence* must be one of the following constants defined in **<unistd.h>**:

SEEK_SET
SEEK_CUR
SEEK_END

If *whence* is **SEEK_SET**, the seek pointer is set to *offset* bytes. If *whence* is **SEEK_CUR**, the seek pointer is set to its current location plus *offset*. If *whence* is **SEEK_END**, the seek pointer is set to the size of the file plus *offset*.

Some devices are incapable of seeking. The value of the seek pointer associated with such a device is undefined.

The obsolete function **tell(fd)** is equivalent to **lseek(fd, 0L, SEEK_CUR)**.

RETURN VALUES

On success, **lseek()** returns the seek pointer location as measured in bytes from the beginning of the file. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

lseek() will fail and the seek pointer will remain unchanged if:

EBADF	*fd* is not an open file descriptor.
EINVAL	*whence* is not a proper value.
	The seek operation would result in an illegal file offset value for the file (for example, a negative file offset for a file other than a character special file).
ESPIPE	*fd* is associated with a pipe or a socket.

SEE ALSO

dup(2V), **open**(2V)

NOTES

Seeking far beyond the end of a file, then writing, may create a gap or "hole", which occupies no physical space and reads as zeros.

The constants **L_SET**, **L_INCR**, and **L_XTND** are provided as synonyms for **SEEK_SET**, **SEEK_CUR**, and **SEEK_END**, respectively for backward compatibility but they will disappear in a future release. It is unlikely that the underlying constants 0, 1 and 2 will ever change.

NAME

mctl – memory management control

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mman.h>

int mctl(addr, len, function, arg)
caddr_t addr;
size_t len;
int function;
void *arg;
```

DESCRIPTION

mctl() applies a variety of control functions over pages identified by the mappings established for the address range [*addr*, *addr* + *len*). The function to be performed is identified by the argument *function*. Legitimate functions are defined in **<sys/mman.h>** as follows.

MC_LOCK
: Lock the pages in the range in memory. This function is used to support **mlock**(3). See the **mlock**(3) description for semantics and usage. *arg* is ignored, but must have the value 0.

MC_LOCKAS
: Lock the pages in the address space in memory. This function is used to support **mlockall**(3). See the **mlockall**(3) description for semantics and usage. *addr* and *len* are ignored but must be 0. *arg* is an integer built from the flags:

```
#define MCL_CURRENT    0x1    /* lock current mappings */
#define MCL_FUTURE     0x2    /* lock future mappings */
```

MC_SYNC
: Synchronize the pages in the range with their backing storage. Optionally invalidate cache copies. This function is used to support **msync**(3). See the **msync**(3) description for semantics and usage. *arg* is used to represent the *flags* argument to **msync**(3).

MC_UNLOCK
: Unlock the pages in the range. This function is used to support **mlock**(3). See the **mlock**(3) description for semantics and usage. *arg* is ignored and must have the value 0.

MC_UNLOCKAS
: Remove address space memory lock, and locks on all current mappings. This function is used to support **mlockall**(3). *addr* and *len* must have the value 0. *arg* is ignored and must have the value 0.

RETURN VALUES

mctl() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EAGAIN
: *function* was **MC_LOCK** or **MC_LOCKAS** and some or all of the memory identified by the operation could not be locked due to insufficient system resources.

EINVAL
: *addr* was not a multiple of the page size as returned by **getpagesize**(2).

: *addr* and/or *len* did not have the value 0 when **MC_LOCKAS** or **MC_UNLOCKAS** were specified.

: *arg* was not valid for the function specified.

ENOMEM
: Addresses in the range [*addr*, *addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

EPERM
: The process's effective user ID was not super-user and one of **MC_LOCK**, **MC_LOCKAS**, **MC_UNLOCK**, or **MC_UNLOCKAS** was specified.

SEE ALSO

madvise(3), **mlock**(3), **mlockall**(3), **mmap**(2), **msync**(3)

NAME

mincore – determine residency of memory pages

SYNOPSIS

int mincore(addr, len, vec)
caddr_t addr; int len; result char *vec;

DESCRIPTION

mincore() returns the primary memory residency status of pages in the address space covered by mappings in the range [*addr*, *addr* + *len*). The status is returned as a char-per-page in the character array referenced by **vec* (which the system assumes to be large enough to encompass all the pages in the address range). The least significant bit of each character is set to 1 to indicate that the referenced page is in primary memory, 0 if it is not. The settings of other bits in each character is undefined and may contain other information in the future.

RETURN VALUES

mincore() returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

mincore() will fail if:

EFAULT	A part of the buffer pointer to by *vec* is out-of-range or otherwise inaccessible.
EINVAL	*addr* is not a multiple of the page size as returned by **getpagesize**(2).
ENOMEM	Addresses in the range [*addr*, *addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

SEE ALSO

mmap(2)

NAME

mkdir – make a directory file

SYNOPSIS

int mkdir(path, mode)
char *path;
int mode;

SYSTEM V SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

int mkdir(path, mode)
char *path;
mode_t mode;

DESCRIPTION

mkdir( ) creates a new directory file with name *path*. The mode mask of the new directory is initialized from *mode*.

The low-order 9 bits of *mode* (the file access permissions) are modified such that all bits set in the process's file mode creation mask are cleared (see **umask**(2V)).

The set-GID bit of *mode* is ignored. The set-GID bit of the new file is inherited from that of the parent directory.

The directory's owner ID is set to the process's effective user ID.

The directory's group ID is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2V)) and the set-GID bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

Upon successful completion, **mkdir( )** marks for update the **st_atime**, **st_ctime**, and **st_mtime** fields of the directory (see **stat**(2V)). The **st_ctime** and **st_mtime** fields of the directory's parent directory are also marked for update.

RETURN VALUES

mkdir( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

mkdir( ) will fail and no directory will be created if:

EACCES	Search permission is denied for a component of the path prefix of *path*.
	Write permission is denied on the parent directory of the directory to be created.
EDQUOT	The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
	The new directory cannot be created because the user's quota of disk blocks on the file system which will contain the directory has been exhausted.
	The user's quota of inodes on the file system on which the file is being created has been exhausted.
EEXIST	The file referred to by *path* exists.
EFAULT	*path* points outside the process's allocated address space.

EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
EMLINK	The link count of the parent directory would exceed {LINK_MAX} (see **pathconf**(2V)).
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix of *path* does not exist.
ENOSPC	The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.
	The new directory cannot be created because there is no space left on the file system which will contain the directory.
	There are no free inodes on the file system on which the file is being created.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EROFS	*path* The parent directory of the directory to be created resides on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to a null pathname.

SEE ALSO

chmod(2V), **mount**(2V), **rmdir**(2V), **stat**(2V), **umask**(2V)

NAME

mknod, mkfifo – make a special file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int mknod(path, mode, dev)
char *path;
int mode, dev;

int mkfifo(path, mode)
char *path;
mode_t mode;
```

DESCRIPTION

mknod() creates a new file named by the path name pointed to by *path*. The mode of the new file (including file type bits) is initialized from *mode*. The values of the file type bits which are permitted are:

```
#define S_IFCHR   0020000   /* character special */
#define S_IFBLK   0060000   /* block special */
#define S_IFREG   0100000   /* regular */
#define S_IFIFO   0010000   /* FIFO special */
```

Values of *mode* other than those above are undefined and should not be used.

The access permissions of the mode are modified by the process's mode mask (see **umask**(2V)).

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2V)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

If *mode* indicates a block or character special file, *dev* is a configuration dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored.

mknod() may be invoked only by the super-user for file types other than FIFO special.

mkfifo() creates a new FIFO special file named by the pathname pointed to by *path*. The access permissions of the new FIFO are initialized from *mode*. The access permissions of *mode* are modified by the process's file creation mask, see **umask**(2V). Bits in *mode* other than the access permissions are ignored.

The FIFO's owner ID is set to the process's effective user ID. The FIFO's group ID is set to the group ID of the directory in which the FIFO is being created or to the process's effective group ID.

Upon successful completion, the **mkfifo()** function marks for update the **st_atime**, **st_ctime**, and **st_mtime** fields of the file. Also, the **st_ctime** and **st_mtime** fields of the directory that contains the new entry are marked for update.

RETURN VALUES

mknod() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

mkfifo() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error. No FIFO is created.

ERRORS

mknod( ) fails and the file mode remains unchanged if:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EDQUOT	The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
EDQUOT	The user's quota of inodes on the file system on which the node is being created has been exhausted.
EEXIST	The file referred to by *path* exists.
EFAULT	*path* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
EISDIR	The specified *mode* would have created a directory.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix of *path* does not exist.
ENOSPC	The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.
ENOSPC	There are no free inodes on the file system on which the file is being created.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EPERM	An attempt was made to create a file of type other than FIFO special and the process's effective user ID is not super-user.
EROFS	The file referred to by *path* resides on a read-only file system.

mkfifo( ) may set **errno** to:

EACCES	A component of the path prefix denies search permission.
EEXIST	The named file already exists.
ENAMETOOLONG	The length of the path string exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix does not exist.
	path points to an empty string.
ENOSPC	The directory that would contain the new file cannot be extended.
	The file system is out of file allocation resources.
ENOTDIR	A component of the path prefix is not a directory.
EROFS	The named file resides on a read-only file system.

SEE ALSO

chmod(2V), **execve**(2V), **pipe**(2V), **stat**(2V), **umask**(2V), **write**(2V)

NAME

mmap – map pages of memory

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mman.h>

caddr_t mmap(addr, len, prot, flags, fd, off)
caddr_t addr;
size_t len;
int prot, flags, fd;
off_t off;
```

DESCRIPTION

mmap() establishes a mapping between the process's address space at an address *pa* for *len* bytes to the memory object represented by *fd* at *off* for *len* bytes. The value of *pa* is an implementation-dependent function of the parameter *addr* and values of *flags*, further described below. A successful **mmap()** call returns *pa* as its result. The address ranges covered by [*pa*, *pa* + *len*) and [*off*, *off* + *len*) must be legitimate for the *possible* (not necessarily current) address space of a process and the object in question, respectively.

The mapping established by **mmap()** replaces any previous mappings for the process's pages in the range [*pa*, *pa* + *len*).

close(2V) does not unmap pages of the object referred to by a descriptor. Use **munmap**(2) to remove a mapping.

The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the pages being mapped. The protection options are defined in **<sys/mman.h>** as:

```
#define PROT_READ     0x1     /* page can be read */
#define PROT_WRITE    0x2     /* page can be written */
#define PROT_EXEC     0x4     /* page can be executed */
#define PROT_NONE     0x0     /* page can not be accessed */
```

Not all implementations literally provide all possible combinations. **PROT_WRITE** is often implemented as **PROT_READ|PROT_WRITE** and **PROT_EXEC** as **PROT_READ|PROT_EXEC**. However, no implementation will permit a write to succeed where **PROT_WRITE** has not been set. The behavior of **PROT_WRITE** can be influenced by setting **MAP_PRIVATE** in the *flags* parameter, described below.

The parameter *flags* provides other information about the handling of the mapped pages. The options are defined in **<sys/mman.h>** as:

```
#define MAP_SHARED    1       /* Share changes */
#define MAP_PRIVATE   2       /* Changes are private */
#define MAP_TYPE      0xf     /* Mask for type of mapping */
#define MAP_FIXED     0x10    /* Interpret addr exactly */
```

MAP_SHARED and **MAP_PRIVATE** describe the disposition of write references to the memory object. If **MAP_SHARED** is specified, write references will change the memory object. If **MAP_PRIVATE** is specified, the initial write reference will create a private copy of the memory object page and redirect the mapping to the copy. The mapping type is retained across a **fork**(2V).

MAP_FIXED informs the system that the value of *pa* must be *addr*, exactly. The use of **MAP_FIXED** is discouraged, as it may prevent an implementation from making the most effective use of system resources.

When **MAP_FIXED** is not set, the system uses *addr* as a hint in an implementation-defined manner to arrive at *pa*. The *pa* so chosen will be an area of the address space which the system deems suitable for a mapping of *len* bytes to the specified object. All implementations interpret an *addr* value of zero as granting the system complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the system selects a value for *pa*, it will never place a mapping at address 0, nor will it replace any extant mapping, nor map into areas considered part of the potential data or stack ''segments''.

The parameter *off* is constrained to be aligned and sized according to the value returned by **getpagesize** (2). When **MAP_FIXED** is specified, the parameter *addr* must also meet these constraints. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system will include in any mapping operation any partial page specified by the range [*pa, pa* + *len*).

mmap() allows [*pa, pa* + *len*) to extend beyond the end of the object, both at the time of the **mmap()** and while the mapping persists, for example if the file was created just prior to the **mmap()** and has no contents, or if the file is truncated. Any reference to addresses beyond the end of the object, however, will result in the delivery of a **SIGBUS** signal.

The system will always zero-fill any partial page at the end of an object. Further, the system will never write out any modified portions of the last page of an object which are beyond its end. References to whole pages following the end of an object will result in a **SIGBUS** signal. **SIGBUS** may also be delivered on various filesystem conditions, including quota exceeded errors.

If the process calls **mlockall**(3) with the **MCL_FUTURE** flag, the pages mapped by all future calls to **mmap()** will be locked in memory. In this case, if not enough memory could be locked, **mmap()** fails and sets **errno** to EAGAIN.

RETURN VALUES

mmap() returns the address at which the mapping was placed (*pa*) on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES — *fd* was not open for read and **PROT_READ** or **PROT_EXEC** were specified.

fd was not open for write and **PROT_WRITE** was specified for a **MAP_SHARED** type mapping.

EAGAIN — Some or all of the mapping could not be locked in memory.

EBADF — *fd* was not open.

EINVAL — The arguments *addr* (if **MAP_FIXED** was specified) and *off* were not multiples of the page size as returned by **getpagesize** (2).

The **MAP_TYPE** field in *flags* was invalid (neither **MAP_PRIVATE** nor **MAP_SHARED**).

ENODEV — *fd* refered to an object for which **mmap()** is meaningless, such as a terminal.

ENOMEM — **MAP_FIXED** was specified, and the range [*addr, addr* + *len*) exceeded that allowed for the address space of a process.

MAP_FIXED was not specified and there was insufficient room in the address space to effect the mapping.

ENXIO — Addresses in the range [*off, off* + *len*) are invalid for *fd*.

SEE ALSO

fork(2V), **getpagesize**(2), **mprotect**(2), **munmap**(2), **mlockall**(3)

NAME

mount – mount file system

SYNOPSIS

```
#include <sys/mount.h>

int mount(type, dir, M_NEWTYPE | flags, data)
char *type;
char *dir;
int flags;
caddr_t data;
```

SYSTEM V SYNOPSIS

```
int mount(spec, dir, rdonly)
char *spec;
char *dir;
int rdonly;
```

DESCRIPTION

mount() attaches a file system to a directory. After a successful return, references to directory *dir* will refer to the root directory on the newly mounted file system. *dir* is a pointer to a null-terminated string containing a path name. *dir* must exist already, and must be a directory. Its old contents are inaccessible while the file system is mounted.

mount() may be invoked only by the super-user.

The *flags* argument is constructed by the logical OR of the following bits (defined in **<sys/mount.h>**):

M_RDONLY mount filesystem read-only.

M_NOSUID ignore set-uid bit on execution.

M_NEWTYPE this flag must always be set.

M_GRPID use BSD file-creation semantics (see **open**(2V)).

M_REMOUNT change options on an existing mount.

M_NOSUB disallow mounts beneath this filesystem.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

The *type* string indicates the type of the filesystem. *data* is a pointer to a structure which contains the type specific arguments to mount. Below is a list of the filesystem types supported and the type specific arguments to each:

```
4.2
    struct ufs_args {
            char        *fspec;       /* Block special file to mount */
    };
"lo"
    struct lo_args {
            char        *fsdir;       /* Pathname of directory to mount */
    };
"nfs"
    #include            <nfs/nfs.h>
    #include            <netinet/in.h>
    struct nfs_args {
            struct sockaddr_in  *addr; /* file server address */
            fhandle_t   *fh;          /* File handle to be mounted */
            int         flags;        /* flags */
```

```
            int       wsize;      /* write size in bytes */
            int       rsize;      /* read size in bytes */
            int       timeo;      /* initial timeout in .1 secs */
            int       retrans;    /* times to retry send */
            char      *hostname;  /* server's hostname */
            int       acregmin;   /* attr cache file min secs */
            int       acregmax;   /* attr cache file max secs */
            int       acdirmin;   /* attr cache dir min secs */
            int       acdirmax;   /* attr cache dir max secs */
            char      *netname;   /* server's netname */
    };
rfs
    struct rfs_args {
            char      *rmtfs      /* name of remote resource */
            struct token {
                      int         t_id;/* token id */
                      char        t_uname[64];/* domain.machine name */
            }         *token;     /* Identifier of remote machine */
    };
```

SYSTEM V DESCRIPTION

mount( ) requests that a file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *spec* and *dir* point to path names. When **mount**( ) succeeds, subsequent references to the file named by *dir* refer to the root directory on the mounted file system.

The **M_RDONLY** bit of *rdonly* is used to control write permission on the mounted file system. If the bit is set, writing is not allowed. Otherwise, writing is permitted according to the access permissions of individual files.

RETURN VALUES

mount( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *dir*.
EBUSY	Another process currently holds a reference to *dir*.
EFAULT	*dir* points outside the process's allocated address space.
ELOOP	Too many symbolic links were encountered in translating the path name of *dir*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENODEV	The file system type specified by *type* is not valid or is not configured into the system.
ENOENT	A component of *dir* does not exist.
ENOTDIR	The file named by *dir* is not a directory.
EPERM	The caller is not the super-user.

For a 4.2 file system, **mount**( ) fails when one of the following occurs:

EACCES	Search permission is denied for a component of the path prefix of *fspec*.

EFAULT	*fspec* points outside the process's allocated address space.
EINVAL	The super block for the file system had a bad magic number or an out of range block size.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating the path name of *fspec*.
EMFILE	No space remains in the mount table.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of *fspec* does not exist.
ENOMEM	Not enough memory was available to read the cylinder group information for the file system.
ENOTBLK	*fspec* is not a block device.
ENOTDIR	A component of the path prefix of *fspec* is not a directory.
ENXIO	The major device number of *fspec* is out of range (this indicates no device driver exists for the associated hardware).

SYSTEM V ERRORS

EBUSY	The device referred to by *spec* is currently mounted.
	There are no more mount table entries.
ENOENT	The file referred to by *spec* or *dir* does not exist.
ENOTBLK	*spec* is not a block special device.
ENOTDIR	A component of the path prefix of *dir* or *spec* is not a directory.
ENXIO	The device referred to by *spec* does not exist.

SEE ALSO

unmount(2V), **open**(2V), **lofs**(4S), **fstab**(5), **mount**(8)

BUGS

Some of the error codes need translation to more obvious messages.

NAME

mprotect – set protection of memory mapping

SYNOPSIS

#include <sys/mman.h>

mprotect(addr, len, prot)
caddr_t addr;
int len, prot;

DESCRIPTION

mprotect( ) changes the access protections on the mappings specified by the range [*addr*, *addr* + *len*) to be that specified by *prot*. Legitimate values for *prot* are the same as those permitted for **mmap**(2).

RETURN VALUES

mprotect( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES *prot* specifies a protection which violates the access permission the process has to the underlying memory object.

EINVAL *addr* is not a multiple of the page size as returned by **getpagesize**(2).

ENOMEM Addresses in the range [*addr*, *addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

When **mprotect( )** fails for reasons other than EINVAL, the protections on some of the pages in the range [*addr*, *addr* + *len*) will have been changed. If the error occurs on some page at address *addr2*, then the protections of all whole pages in the range [*addr*, *addr2*) have been modified.

SEE ALSO

getpagesize(2), **mmap**(2)

NAME

msgctl – message control operations

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;

DESCRIPTION

msgctl( ) provides a variety of message control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in **intro**(2).

IPC_SET Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

msg_perm.uid
msg_perm.gid
msg_perm.mode /* only low 9 bits */
msg_qbytes

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*. Only super-user can raise the value of **msg_qbytes**.

IPC_RMID Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid.*

In the **msgop**(2) and **msgctl**(2) system call descriptions, the permission required for an operation is given as "[token]", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a msqid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **msg_perm.[c]uid** in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process matches **msg_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process does not match **msg_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

RETURN VALUES

msgctl() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES *cmd* is equal to IPC_STAT and **[READ]** operation permission is denied to the calling process (see **intro**(2)).

EFAULT *buf* points to an illegal address.

EINVAL *msqid* is not a valid message queue identifier.

cmd is not a valid command.

EPERM *cmd* is equal to **IPC_RMID** or **IPC_SET**. The effective user ID of the calling process is neither super-user, nor the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*.

cmd is equal to IPC_SET, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process is not equal to that of super-user.

SEE ALSO

intro(2), **msgget**(2), **msgop**(2)

NAME

msgget – get message queue

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key, msgflg)
key_t key;
int msgflg;
```

DESCRIPTION

msgget() returns the message queue identifier associated with **key**.

A message queue identifier and associated message queue and data structure (see **intro**(2)) are created for **key()** if one of the following is true:

- *key* is equal to **IPC_PRIVATE**.
- *key* does not already have a message queue identifier associated with it, and (*msgflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

- **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **msg_perm.mode** are set equal to the low-order 9 bits of *msgflg*.
- **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** are set equal to 0.
- **msg_ctime** is set equal to the current time.
- **msg_qbytes** is set equal to the system-wide standard value of the maximum number of bytes allowed on a message queue.

A message queue identifier (msqid) is a unique positive integer created by a **msgget**(2) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as **msqid_ds()** and contains the following members:

```
struct   ipc_perm msg_perm;  /* operation permission struct */
ushort   msg_qnum;           /* number of msgs on q */
ushort   msg_qbytes;         /* max number of bytes on q */
ushort   msg_lspid;          /* pid of last msgsnd operation */
ushort   msg_lrpid;          /* pid of last msgrcv operation */
time_t   msg_stime;          /* last msgsnd time */
time_t   msg_rtime;          /* last msgrcv time */
time_t   msg_ctime;          /* last change time */
                             /* Times measured in secs since */
                             /* 00:00:00 GMT, Jan. 1, 1970 */
```

msg_perm() is an ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort   cuid;               /* creator user id */
ushort   cgid;               /* creator group id */
ushort   uid;                /* user id */
ushort   gid;                /* group id */
ushort   mode;               /* r/w permission */
```

msg_qnum is the number of messages currently on the queue. **msg_qbytes** is the maximum number of bytes allowed on the queue. **msg_lspid** is the process ID of the last process that performed a **msgsnd** operation. **msg_lrpid** is the process ID of the last process that performed a **msgrcv** operation. **msg_stime**

is the time of the last *msgsnd* operation, **msg_rtime** is the time of the last *msgrcv* operation, and **msg_ctime** is the time of the last **msgctl**(2) operation that changed a member of the above structure.

RETURN VALUES

msgget() returns A non-negative message queue identifier on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EACCES — A message queue identifier exists for **key**, but operation permission (see **intro**(2)) as specified by the low-order 9 bits of **msgflg** would not be granted.

EEXIST — A message queue identifier exists for **key()** but ((**msgflg** & **IPC_CREAT**) & (**msgflg** & **IPC_EXCL**)) is "true".

ENOENT — A message queue identifier does not exist for **key()** and (**msgflg** & **IPC_CREAT**) is "false".

ENOSPC — A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.

SEE ALSO

intro(2), **msgctl**(2), **msgop**(2)

NAME

msgop, msgsnd, msgrcv – message operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv(msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

DESCRIPTION

msgsnd() is used to send a message to the queue associated with the message queue identifier specified by *msqid*. **[WRITE]** (see **msgctl**(2)) *msgp* points to a structure containing the message. This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[1];   /* message text */
```

mtype is a positive integer that can be used by the receiving process for message selection (see **msgrcv()** below). *mtext* is any text of length *msgsz* bytes. *msgsz* can range from 0 to a system-imposed maximum.

msgflg specifies the action to be taken if one or more of the following are true:

- The number of bytes already on the queue is equal to *msg_qbytes* (see **intro**(2)).
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- If **(msgflg & IPC_NOWAIT)** is "true", the message will not be sent and the calling process will return immediately.
- If **(msgflg & IPC_NOWAIT)** is "false", the calling process will suspend execution until one of the following occurs:
  - The condition responsible for the suspension no longer exists, in which case the message is sent.
  - *msqid* is removed from the system (see **msgctl**(2)). When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.
  - The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in **signal**(3V).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see **intro**(2)).

- *msg_qnum* is incremented by 1.
- *msg_lspid* is set equal to the process ID of the calling process.
- *msg_stime* is set equal to the current time.

msgrcv() reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. **[READ]** This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[1];   /* message text */
```

mtype is the received message's type as specified by the sending process. *mtext* is the text of the message. *msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (**msgflg & MSG_NOERROR**) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

msgtyp specifies the type of message requested as follows:

- If *msgtyp* is equal to 0, the first message on the queue is received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

msgflg specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If (**msgflg & IPC_NOWAIT**) is "true", the calling process will return immediately with a return value of −1 and **errno** set to ENOMSG.
- If (**msgflg & IPC_NOWAIT**) is "false", the calling process will suspend execution until one of the following occurs:
  - A message of the desired type is placed on the queue.
  - *msqid* is removed from the system. When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.
  - The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in **signal**(3V).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see **intro**(2)).

- *msg_qnum* is decremented by 1.
- *msg_lrpid* is set equal to the process ID of the calling process.
- *msg_rtime* is set equal to the current time.

RETURN VALUES

msgsnd( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

msgrcv( ) returns the number of bytes actually placed into *mtext* on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

msgsnd( ) will fail and no message will be sent if one or more of the following are true:

EACCES	Operation permission is denied to the calling process (see **intro**(2)).
EAGAIN	The message cannot be sent for one of the reasons cited above and (**msgflg & IPC_NOWAIT**) is "true".
EFAULT	*msgp* points to an illegal address.
EIDRM	The message queue referred to by *msqid* was removed from the system.
EINTR	The call was interrupted by the delivery of a signal.

EINVAL *msqid* is not a valid message queue identifier.

mtype is less than 1.

msgsz is less than zero or greater than the system-imposed limit.

msgrcv( ) will fail and no message will be received if one or more of the following are true:

E2BIG *mtext* is greater than *msgsz* and (**msgflg** & **MSG_NOERROR**) is "false".

EACCES Operation permission is denied to the calling process.

EFAULT *msgp* points to an illegal address.

EIDRM The message queue referred to by *msqid* was removed from the system.

EINTR The call was interrupted by the delivery of a signal.

EINVAL *msqid* is not a valid message queue identifier.

msgsz is less than 0.

ENOMSG The queue does not contain a message of the desired type and (**msgtyp** & **IPC_NOWAIT**) is "true".

SEE ALSO

intro(2), **msgctl**(2), **msgget**(2), **signal**(3V)

NAME

msync – synchronize memory with physical storage

SYNOPSIS

#include <sys/mman.h>

int msync(addr, len, flags)
caddr_t addr;
int len, flags;

DESCRIPTION

msync() writes all modified copies of pages over the range [*addr*, *addr* + *len*) to their permanent storage locations. **msync()** optionally invalidates any copies so that further references to the pages will be obtained by the system from their permanent storage locations.

Values for *flags* are defined in **<sys/mman.h>** as:

```
#define MS_ASYNC        0x1     /* Return immediately */
#define MS_INVALIDATE   0x2     /* Invalidate mappings */
```

and are used to control the behavior of **msync()**. One or more flags may be specified in a single call.

MS_ASYNC returns **msync()** immediately once all I/O operations are scheduled; normally, **msync()** will not return until all I/O operations are complete. **MS_INVALIDATE** invalidates all cached copies of data from memory objects, requiring them to be re-obtained from the object's permanent storage location upon the next reference.

msync() should be used by programs which require a memory object to be in a known state, for example in building transaction facilities.

RETURN VALUES

msync() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL *addr* is not a multiple of the current page size.

len is negative.

One of the flags MS_ASYNC or MS_INVALID is invalid.

EIO An I/O error occurred while reading from or writing to the file system.

ENOMEM Addresses in the range [*addr*, *addr* + *len*) are outside the valid range for the address space of a process.

NAME

munmap – unmap pages of memory.

SYNOPSIS

#include <sys/mman.h>

int munmap(addr, len)
caddr_t addr;
int len;

DESCRIPTION

munmap() removes the mappings for pages in the range [*addr*, *addr* + *len*). Further references to these pages will result in the delivery of a **SIGSEGV** signal to the process, unless these pages are considered part of the "data" or "stack" segments.

brk() and **mmap()** often perform implicit **munmap**'s.

RETURN VALUES

munmap() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL *addr* is not a multiple of the page size as returned by **getpagesize**(2).

Addresses in the range [*addr*, *addr* + *len*) are outside the valid range for the address space of a process.

SEE ALSO

brk(2), **getpagesize**(2), **mmap**(2)

NAME

nfssvc, async_daemon – NFS daemons

SYNOPSIS

nfssvc (sock)
int sock;

async_daemon()

DESCRIPTION

nfssvc() starts an NFS daemon listening on socket *sock*. The socket must be **AF_INET**, and **SOCK_DGRAM** (protocol **UDP/IP**). The system call will return only if the socket is invalid.

async_daemon() implements the NFS daemon that handles asynchronous I/O for an NFS client. This system call never returns.

Both system calls result in kernel-only processes with user memory discarded.

SEE ALSO

mountd(8C)

BUGS

There should be a way to dynamically create kernel-only processes instead of having to make system calls from userland to simulate this.

NAME

open – open or create a file for reading or writing

SYNOPSIS

```
#include <fcntl.h>

int open(path, flags[ , mode ] )
char *path;
int flags;
int mode;
```

SYSTEM V SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(path, flags[ , mode ] )
char *path;
int flags;
mode_t mode;
```

DESCRIPTION

path points to the pathname of a file. **open**( ) opens the named file for reading and/or writing, as specified by the *flags* argument, and returns a descriptor for that file. The *flags* argument may indicate the file is to be created if it does not already exist (by specifying the **O_CREAT** flag), in which case the file is created with mode *mode* as described in **chmod**(2V) and modified by the process' umask value (see **umask**(2V)). If the path is an empty string, the kernel maps this empty pathname to '.', the current directory. *flags* values are constructed by ORing flags from the following list (one and only one of the first three flags below must be used):

O_RDONLY　Open for reading only.

O_WRONLY　Open for writing only.

O_RDWR　Open for reading and writing.

O_NDELAY　When opening a FIFO (named pipe – see **mknod**(2V)) with **O_RDONLY** or **O_WRONLY** set:

If **O_NDELAY** is set:

An **open**( ) for reading-only returns without delay. An **open**( ) for writing-only returns an error if no process currently has the file open for reading.

If **O_NDELAY** is clear:

A call to **open**( ) for reading-only blocks until a process opens the file for writing. A call to **open**( ) for writing-only blocks until a process opens the file for reading.

When opening a file associated with a communication line:

If **O_NDELAY** is set:

A call to **open**( ) returns without waiting for carrier.

If **O_NDELAY** is clear:

A call to **open**( ) blocks until carrier is present.

O_NOCTTY　When this flag is set, and *path* refers to a terminal device, **open**( ) prevents the terminal device from becoming the controlling terminal for the process.

O_NONBLOCK　Same as **O_NDELAY** above.

O_SYNC When opening a regular file, this flag affects subsequent writes. If set, each **write**(2V) will wait for both the file data and file status to be physically updated.

O_APPEND If set, the seek pointer will be set to the end of the file prior to each write.

O_CREAT If the file exists, this flag has no effect. Otherwise, the file is created, and the owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2V)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

The low-order 12 bits of the file mode are set to the value of *mode*, modified as follows (see **creat**(2V)):

- All bits set in the file mode creation mask of the process are cleared. See **umask**(2V).
- The "save text image after execution" bit of the mode is cleared. See **chmod**(2V).
- The "set group ID on execution" bit of the mode is cleared if the effective user ID of the process is not super-user and the process is not a member of the group of the created file.

O_TRUNC If the file exists and is a regular file, and the file is successfully opened **O_RDWR** or **O_WRONLY**, its length is truncated to zero and the mode and owner are unchanged. **O_TRUNC** has no effect on FIFO special files or directories.

O_EXCL If **O_EXCL** and **O_CREAT** are set, **open**( ) will fail if the file exists. This can be used to implement a simple exclusive access locking mechanism.

The seek pointer used to mark the current position within the file is set to the beginning of the file.

The new descriptor is set to remain open across **execve**(2V) system calls; see **close**(2V) and **fcntl**(2V).

There is a system enforced limit on the number of open file descriptors per process, whose value is returned by the **getdtablesize**(2) call.

If **O_CREAT** is set and the file did not previously exist, upon successful completion, **open**( ) marks for update the **st_atime**, **st_ctime**, and **st_mtime** fields of the file and the **st_ctime** and **st_mtime** fields of the parent directory.

If **O_TRUNC** is set and the file previously existed, upon successful completion, **open**( ) marks for update the **st_ctime** and **st_mtime** fields of the file.

SYSTEM V DESCRIPTION

If *path* points to an empty string an error results.

The flags above behave as described, with the following exception:

If the **O_NDELAY** or **O_NONBLOCK** flag is set on a call to **open**( ), the corresponding flag is set for that file descriptor (see **fcntl**(2V)) and subsequent reads and writes to that descriptor will not block (see **read**(2V) and **write**(2V)).

RETURN VALUES

open( ) returns a non-negative file descriptor on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES
: Search permission is denied for a component of the path prefix of *path*.

The file referred to by *path* does not exist, **O_CREAT** is specified, and the directory in which it is to be created does not permit writing.

O_TRUNC is specified and write permission is denied for the file named by *path*.

The required permissions (for reading and/or writing) are denied for the file named by *path*.

EDQUOT
: The file does not exist, **O_CREAT** is specified, and the directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.

The file does not exist, **O_CREAT** is specified, and the user's quota of inodes on the file system on which the file is being created has been exhausted.

EEXIST
: **O_EXCL** and **O_CREAT** were both specified and the file exists.

EFAULT
: *path* points outside the process's allocated address space.

EINTR
: A signal was caught during the **open()** system call.

EIO
: A hangup or error occurred during a STREAMS **open()**.

An I/O error occurred while reading from or writing to the file system.

EISDIR
: The named file is a directory, and the arguments specify it is to be opened for writing.

ELOOP
: Too many symbolic links were encountered in translating *path*.

EMFILE
: The system limit for open file descriptors per process has already been reached.

ENAMETOOLONG
: The length of the path argument exceeds {PATH_MAX}.

A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).

ENFILE
: The system file table is full.

ENOENT
: **O_CREAT** is not set and the named file does not exist.

A component of the path prefix of *path* does not exist.

ENOSPC
: The file does not exist, **O_CREAT** is specified, and the directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

The file does not exist, **O_CREAT** is specified, and there are no free inodes on the file system on which the file is being created.

ENOSR
: A *stream* could not be allocated.

ENOTDIR
: A component of the path prefix of *path* is not a directory.

ENXIO
: **O_NDELAY** is set, the named file is a FIFO, **O_WRONLY** is set, and no process has the file open for reading.

The file is a character special or block special file, and the associated device does not exist.

O_NONBLOCK is set, the named file is a FIFO, **O_WRONLY** is set, and no process has the file open for reading.

A STREAMS module or driver open routine failed.

EOPNOTSUPP
: An attempt was made to open a socket (not currently implemented).

EROFS The named file does not exist, **O_CREAT** is specified, and the file system on which it is to be created is a read-only file system.

The named file resides on a read-only file system, and the file is to be opened for writing.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT *path* points to an empty string.

SEE ALSO

chmod(2V), **close**(2V), **creat**(2V), **dup**(2V), **fcntl**(2V), **getdtablesize**(2), **getmsg**(2), **lseek**(2V), **mknod**(2V), **mount**(2V), **putmsg**(2), **read**(2V), **umask**(2V) **write**(2V)

NAME

pathconf, fpathconf – query file system related limits and options

SYNOPSIS

```
#include <unistd.h>

long pathconf(path, name)
char *path;
int name;

long fpathconf(fd, name)
int fd, name;
```

DESCRIPTION

pathconf() and **fpathconf()** provide a method for the application to determine the current value of a configurable limit or option that is associated with a file or directory,

For **pathconf()**, *path* points to the pathname of a file or directory. For **fpathconf()**, *fd* is an open file descriptor.

The convention used throughout sections 2 and 3 is that {LIMIT} means that LIMIT is something that can change from file to file (due to multiple file systems on the same machine). The actual value for LIMIT is typically not defined in any header file since it is not invariant. Instead, pathconf must be called to retrieve the value. **pathconf()** understands a list of flags that are named similarly to the value being queried.

The following table lists the name and meaning of each conceptual limit.

Limit	*Meaning*
{LINK_MAX}	Max links to an object.
{MAX_CANON}	Max tty input line size.
{MAX_INPUT}	Max packet a tty can accept at once.
{NAME_MAX}	Max filename length.
{PATH_MAX}	Max pathname length.
{PIPE_BUF}	Pipe buffer size.
{_POSIX_CHOWN_RESTRICTED}	If true only root can chown() files, otherwise anyone may give away files.
{_POSIX_NO_TRUNC}	If false filenames > {NAME_MAX} are truncated, otherwise an error.
{_POSIX_VDISABLE}	A char to use to disable tty special chars.

The following table lists the name of each limit, the flag passed to **pathconf()** to retrieve the value of each variable, and some notes about usage.

Limit	*Pathconf Flag*	*Notes*
{LINK_MAX}	**_PC_LINK_MAX**	1
{MAX_CANON}	**_PC_MAX_CANON**	2
{MAX_INPUT}	**_PC_MAX_INPUT**	2
{NAME_MAX}	**_PC_NAME_MAX**	3,4
{PATH_MAX}	**_PC_PATH_MAX**	4,5
{PIPE_BUF}	**_PC_PIPE_BUF**	6
{_POSIX_CHOWN_RESTRICTED}	**_PC_CHOWN_RESTRICTED**	7,8
{_POSIX_NO_TRUNC}	**_PC_NO_TRUNC**	3,4,8
{_POSIX_VDISABLE}	**_PC_VDISABLE**	2,8

The following notes apply to the entries in the preceding table.

1 If *path* or *fd* refers to a directory, the value returned applies to the directory itself.

2 The behavior is undefined if *path* or *fd* does not refer to a terminal file.

3 If *path* or *fd* refers to a directory, the value returned applies to the file names within the directory.

4 The behavior is undefined if *path* or *fd* does not refer to a directory.

5 If *path* or *fd* refers to a directory, the value returned is the maximum length of a relative pathname when the specified directory is the working directory.

6 If *path* refers to a FIFO, or *fd* refers to a pipe of FIFO, the value returned applies to the referenced object itself. If *path* or *fd* refers to a directory, the value returned applies to any FIFOs that exist or can be created within the directory. If *path* or *fd* refer to any other type of file, the behavior is undefined.

7 If *path* or *fd* refer to a directory, the value returned applies to any files, other than directories, that exist or can be created within the directory.

8 The option in question is a boolean; the return value is 0 or 1.

RETURN VALUES

On success, **pathconf()** and **fpathconf()** return the current variable value for the file or directory. On failure, they return −1 and set **errno** to indicate the error.

If the variable corresponding to *name* has no limit for the path or file descriptor, **pathconf()** and **fpathconf()** return −1 without changing **errno**.

ERRORS

pathconf() and **fpathconf()** may set **errno** to:

EINVAL The value of name is invalid.

For each of the following conditions, if the condition is detected, **pathconf()** fails and sets **errno** to:

EACCES Search permission is denied for a component of the path prefix.

EINVAL The implementation does not support an association of the variable name with the specified file.

ENAMETOOLONG The length of the path argument exceeds {PATH_MAX}.

A pathname component is longer than {NAME_MAX} while {POSIX_NO_TRUNC} is in effect.

ENOENT The named file does not exist.

path points to an empty string.

ENOTDIR A component of the path prefix is not a directory.

For each of the following conditions, if the condition is detected, **fpathconf()** fails and sets **errno** to:

EBADF

The *fd* argument is not a valid file descriptor.

EINVAL The implementation does not support an association of the variable name with the specified file.

NAME

pipe – create an interprocess communication channel

SYNOPSIS

```
int pipe(fd)
int fd[2];
```

DESCRIPTION

The **pipe()** system call creates an I/O mechanism called a pipe and returns two file descriptors, *fd*[0] and *fd*[1]. *fd*[0] is opened for reading and *fd*[1] is opened for writing. The **O_NONBLOCK** flag is clear on both file descriptors (see **open**(2V)). When the pipe is written using the descriptor *fd*[1] up to {PIPE_BUF} (see **sysconf**(2V)) bytes of data are buffered before the writing process is blocked. A read only file descriptor *fd*[0] accesses the data written to *fd*[1] on a FIFO (first-in-first-out) basis.

The standard programming model is that after the pipe has been set up, two (or more) cooperating processes (created by subsequent **fork**(2V) calls) will pass data through the pipe using **read**(2V) and **write**(2V).

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an EOF (end of file).

Pipes are really a special case of the **socketpair**(2) call and, in fact, are implemented as such in the system.

A **SIGPIPE** signal is generated if a write on a pipe with only one end is attempted.

Upon successful completion, **pipe()** marks for update the **st_atime**, **st_ctime**, and **st_mtime** fields of the pipe.

RETURN VALUES

pipe() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EFAULT	The array *fd* is in an invalid area of the process's address space.
EMFILE	Too many descriptors are active.
ENFILE	The system file table is full.

SEE ALSO

sh(1), **fork**(2V), **read**(2V), **socketpair**(2), **write**(2V)

BUGS

Should more than {PIPE_BUF} bytes be necessary in any pipe among a loop of processes, deadlock will occur.

NAME

poll – I/O multiplexing

SYNOPSIS

```
#include <poll.h>

int poll(fds, nfds, timeout)
struct pollfd *fds;
unsigned long nfds;
int timeout;
```

DESCRIPTION

poll() provides users with a mechanism for multiplexing input/output over a set of file descriptors (see **intro**(2)). **poll()** identifies those file descriptors on which a user can send or receive messages, or on which certain events have occurred. A user can receive messages using **read**(2V) or **getmsg**(2) and can send messages using **write**(2V) and **putmsg**(2). Certain **ioctl**(2) calls, such as **I_RECVFD** and **I_SENDFD** (see **streamio**(4)), can also be used to receive and send messages on streams.

fds specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one element for each open file descriptor of interest. The array's elements are **pollfd** structures which contain the following members:

```
int fd;              /* file descriptor */
short events;        /* requested events */
short revents;       /* returned events */
```

where **fd** specifies an open file descriptor and **events** and **revents** are bitmasks constructed by ORing any combination of the following event flags:

POLLIN If the file descriptor refers to a stream, a non-priority or file descriptor passing message (see **I_RECVFD**) is present on the **stream head** read queue. This flag is set even if the message is of zero length. If the file descriptor is not a stream, the file descriptor is readable. In **revents**, this flag is mutually exclusive with **POLLPRI**.

POLLPRI If the file descriptor is a stream, a priority message is present on the **stream head** read queue. This flag is set even if the message is of zero length. If the file descriptor is not a stream, some exceptional condition has occurred. In **revents**, this flag is mutually exclusive with **POLLIN**.

POLLOUT If the file descriptor is a stream, the first downstream write queue in the *stream* is not full. Priority control messages can be sent (see **putmsg**(2)) at any time. If the file descriptor is not a stream, it is writable.

POLLERR If the file descriptor is a stream, an error message has arrived at the **stream head**. This flag is only valid in the **revents** bitmask; it is not used in the **events** field.

POLLHUP If the file descriptor is a stream, a hangup has occurred on the *stream*. This event and **POLLOUT** are mutually exclusive; a *stream* can never be writable if a hangup has occurred. However, this event and **POLLIN** or **POLLPRI** are not mutually exclusive. This flag is only valid in the **revents** bitmask; it is not used in the *events* field.

POLLNVAL The specified **fd** value does not specify an open file descriptor. This flag is only valid in the **revents** field; it is not used in the **events** field.

For each element of the array pointed to by *fds*, **poll()** examines the given file descriptor for the **event**(s) specified in **events**. The number of file descriptors to be examined is specified by *nfds*. If *nfds* exceeds the system limit of open files (see **getdtablesize**(2)), **poll()** will fail.

If the value **fd** is less than zero, **events** is ignored and **revents** is set to 0 in that entry on return from **poll()**.

The results of the **poll()** query are stored in the **revents** field in the **pollfd** structure. Bits are set in the **revents** bitmask to indicate which of the requested events are true. If none are true, none of the specified bits is set in **revents** when the **poll()** call returns. The event flags **POLLHUP, POLLERR**, and **POLLNVAL** are always set in **revents** if the conditions they indicate are true; this occurs even though these flags were not present in **events**.

If none of the defined events have occurred on any selected file descriptor, **poll()** waits at least *timeout* milliseconds for an event to occur on any of the selected file descriptors. On a computer where millisecond timing accuracy is not available, *timeout* is rounded up to the nearest legal value available on that system. If the value *timeout* is 0, **poll()** returns immediately. If the value of *timeout* is −1, **poll()** blocks until a requested event occurs or until the call is interrupted. **poll()** is not affected by the **O_NDELAY** flag.

RETURN VALUES

poll() returns a non-negative value on success. A positive value indicates the total number of file descriptors that has been selected (for instance, file descriptors for which the **revents** field is non-zero). 0 indicates the call timed out and no file descriptors have been selected. On failure, **poll()** returns −1 and sets **errno** to indicate the error.

ERRORS

EAGAIN	Allocation of internal data structures failed, but the request should be attempted again.
EFAULT	Some argument points outside the allocated address space.
EINTR	A signal was caught during the **poll()** system call.
EINVAL	The argument *nfds* is less than zero.
	nfds is greater than the system limit of open files.

SEE ALSO

getdtablesize(2), **getmsg**(2), **intro**(2), **ioctl**(2), **putmsg**(2), **read**(2V), **select**(2), **write**(2V), **streamio**(4)

NAME

profil – execution time profile

SYNOPSIS

int profil(buf, bufsiz, offset, scale)
short *buf;
int bufsiz;
void (*offset)();
int scale;

DESCRIPTION

profil( ) enables run-time execution profiling, and reserves a buffer for maintaining raw profiling statistics. *buf* points to an area of core of length *bufsiz* (in bytes). After the call to **profil( )**, the user's program counter (pc) is examined at each clock tick (10 milliseconds on Sun-4 systems, 20 milliseconds on Sun-3 systems); *offset* is subtracted from its value, and the result multiplied by *scale*. If the resulting number corresponds to a word within the buffer, that word is incremented.

scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0xffff gives a 1-to-1 mapping of pc values to words in *buf*; 0x7fff maps each pair of instruction words together. 0x2 maps all instructions onto the beginning of *buf* (producing a non-interrupting core clock).

Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an **execve( )** is executed, but remains on in child and parent both after a **fork( )**. Profiling is turned off if an update in *buf* would cause a memory fault.

RETURN VALUES

profil( ) always succeeds and returns 0.

SEE ALSO

gprof(1), **getitimer**(2), **monitor**(3)

NAME

ptrace – process trace

SYNOPSIS

```
#include <signal.h>
#include <sys/ptrace.h>
#include <sys/wait.h>

ptrace(request, pid, addr, data [ , addr2 ] )
enum ptracereq request;
int pid;
char *addr;
int data;
char *addr2;
```

DESCRIPTION

ptrace( ) provides a means by which a process may control the execution of another process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are five arguments whose interpretation depends on the *request* argument. Generally, *pid* is the process ID of the traced process. A process being traced behaves normally until it encounters some signal whether internally generated like "illegal instruction" or externally generated like "interrupt". See **sigvec**(2) for the list. Then the traced process enters a stopped state and the tracing process is notified using **wait**(2V). When the traced process is in the stopped state, its core image can be examined and modified using **ptrace**( ). If desired, another **ptrace**( ) request can then cause the traced process either to terminate or to continue, possibly ignoring the signal.

Note: several different values of the *request* argument can make **ptrace**( ) return data values — since −1 is a possibly legitimate value, to differentiate between −1 as a legitimate value and −1 as an error code, you should clear the **errno** global error code before doing a **ptrace**( ) call, and then check the value of **errno** afterwards.

The value of the *request* argument determines the precise action of the call:

PTRACE_TRACEME

This request is the only one used by the traced process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

PTRACE_PEEKTEXT
PTRACE_PEEKDATA

The word in the traced process's address space at *addr* is returned. If the instruction and data spaces are separate (for example, historically on a PDP-11), request **PTRACE_PEEKTEXT** indicates instruction space while **PTRACE_PEEKDATA** indicates data space. Otherwise, either request may be used, with equal results; *addr* must be a multiple of 4 on a Sun-4 system. The child must be stopped. The input *data* and *addr2* are ignored.

PTRACE_PEEKUSER

The word of the system's per-process data area corresponding to *addr* is returned. *addr* must be a valid offset within the kernel's per-process data pages. This space contains the registers and other information about the process; its layout corresponds to the *user* structure in the system (see **<sys/user.h>**).

PTRACE_POKETEXT
PTRACE_POKEDATA

The given *data* are written at the word in the process's address space corresponding to *addr*. *addr* must be a multiple of 4 on a Sun-4 system. No useful value is returned. If the instruction and data spaces are separate, request **PTRACE_PEEKTEXT** indicates instruction space while **PTRACE_PEEKDATA** indicates data space. The **PTRACE_POKETEXT** request must be used to write into a process's text space even if the instruction and data spaces are not separate.

PTRACE_POKEUSER
The process's system data are written, as it is read with request **PTRACE_PEEKUSER**. Only a few locations can be written in this way: the general registers, the floating point and status registers, and certain bits of the processor status word.

PTRACE_CONT
The *data* argument is taken as a signal number and the child's execution continues at location *addr* as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If *addr* is (int *)1 then execution continues from where it stopped. *addr* must be a multiple of 4 on a Sun-4 system.

PTRACE_KILL
The traced process terminates, with the same consequences as **exit**(2V).

PTRACE_SINGLESTEP
Execution continues as in request **PTRACE_CONT**; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is **SIGTRAP**. On Sun-3 and Sun386i systems, the status register T-bit is used and just one instruction is executed. This is part of the mechanism for implementing breakpoints. On a Sun-4 system this will return an error since there is no hardware assist for this feature. Instead, the user should insert breakpoint traps in the debugged program with **PTRACE_POKETEXT**.

PTRACE_ATTACH
Attach to the process identified by the *pid* argument and begin tracing it. **PTRACE_ATTACH** causes a **SIGSTOP** to be sent to process *pid*. Process *pid* does not have to be a child of the requestor, but the requestor must have permission to send process *pid* a signal and the effective user IDs of the requesting process and process *pid* must match.

PTRACE_DETACH
Detach the process being traced. Process *pid* is no longer being traced and continues its execution. The *data* argument is taken as a signal number and the process continues at location *addr* as if it had incurred that signal.

PTRACE_GETREGS
The traced process's registers are returned in a structure pointed to by the *addr* argument. The registers include the general purpose registers, the program counter and the program status word. The "regs" structure defined in **<machine/reg.h>** describes the data that are returned.

PTRACE_SETREGS
The traced process's registers are written from a structure pointed to by the *addr* argument. The registers include the general purpose registers, the program counter and the program status word. The "regs" structure defined in **reg.h** describes the data that are set.

PTRACE_GETFPREGS
(Sun-3, Sun-4 and Sun386i systems only) The traced process's FPP status is returned in a structure pointed to by the *addr* argument. The status includes the 68881 (80387 on Sun386i systems) floating point registers and the control, status, and instruction address registers. The "fp_status" structure defined in **reg.** describes the data that are returned. The **fp_state** structure defined in **<machine/fp.h>** describes the data that are returned on a Sun386i system.

PTRACE_SETFPREGS
(Sun-3, Sun-4 and Sun386i systems only) The traced process's FPP status is written from a structure pointed to by the *addr* argument. The status includes the FPP floating point registers and the control, status, and instruction address registers. The "fp_status" structure defined in **reg.h** describes the data that are set. The "fp_state" structure defined in **fp.h** describes the data that are returned on a Sun386i system.

PTRACE_GETFPAREGS
(a Sun-3 system with FPA only) The traced process's FPA registers are returned in a structure pointed to by the *addr* argument. The "fpa_regs" structure defined in **reg.h** describes the data that are returned.

PTRACE_SETFPAREGS
(a Sun-3 system with FPA only) The traced process's FPA registers are written from a structure pointed to by the *addr* argument. The "fpa_regs" structure defined in **reg.h** describes the data that are set.

PTRACE_READTEXT
PTRACE_READDATA
Read data from the address space of the traced process. If the instruction and data spaces are separate, request **PTRACE_READTEXT** indicates instruction space while **PTRACE_READDATA** indicates data space. The *addr* argument is the address within the traced process from where the data are read, the *data* argument is the number of bytes to read, and the *addr2* argument is the address within the requesting process where the data are written.

PTRACE_WRITETEXT
PTRACE_WRITEDATA
Write data into the address space of the traced process. If the instruction and data spaces are separate, request **PTRACE_READTEXT** indicates instruction space while **PTRACE_READDATA** indicates data space. The *addr* argument is the address within the traced process where the data are written, the *data* argument is the number of bytes to write, and the *addr2* argument is the address within the requesting process from where the data are read.

PTRACE_SETWRBKPT
(Sun386i systems only) Set a write breakpoint at location *addr* in the process being traced. Whenever a write is directed to this location a breakpoint will occur and a SIGTRAP signal will be sent to the process. The *data* argument specifies which debug register should be used for the address of the breakpoint and must be in the range 0 through 3, inclusive. The *addr2* argument specifies the length of the operand in bytes, and must be one of 1, 2, or 4.

PTRACE_SETACBKPT
(Sun386i systems only) Set an access breakpoint at location *addr* in the process being traced. When location *addr* is read or written a breakpoint will occur and the process will be sent a SIGTRAP signal. The *data* argument specifies which debug register should be used for the address of the breakpoint and must be in the range 0 through 3, inclusive. The *addr2* argument specifies the length of the operand in bytes, and must be one of 1, 2, or 4.

PTRACE_CLRBKPT
(Sun386i systems only) Clears all break points set with **PTRACE_SETACBKPT** or **PTRACE_SETWRBKPT**.

PTRACE_SYSCALL
Execution continues as in request **PTRACE_CONT**; until the process makes a system call. The process receives a **SIGTRAP** signal and stops. At this point the arguments to the system call may be inspected in the process *user* structure using the **PTRACE_PEEKUSER** request. The system call number is available in place of the 8th argument. Continuing with another **PTRACE_SYSCALL** will stop the process again at the completion of the system call. At this point the result of the system call and error value may be inspected in the process *user* structure.

PTRACE_DUMPCORE
Dumps a core image of the traced process to a file. The name of the file is obtained from the *addr* argument.

As indicated, these calls (except for requests **PTRACE_TRACEME**, **PTRACE_ATTACH** and **PTRACE_DETACH**) can be used only when the subject process has stopped. The **wait**( ) call is used to determine when a process stops; in such a case the "termination" status returned by **wait**( ) has the value **WSTOPPED** to indicate a stop rather than genuine termination.

To forestall possible fraud, **ptrace**( ) inhibits the setUID and setGID facilities on subsequent **execve**(2V) calls. If a traced process calls **execve**( ), it will stop before executing the first instruction of the new image, showing signal **SIGTRAP**.

On the Sun, "word" also means a 32-bit integer.

RETURN VALUES

On success, the value returned by **ptrace**( ) depends on *request* as follows:

PTRACE_PEEKTEXT **PTRACE_PEEKDATA**	The word in the traced process's address space at *addr*.
PTRACE_PEEKUSER	The word of the system's per-process data area corresponding to *addr*.

On failure, these requests return −1 and set **errno** to indicate the error.

For all other values of *request*, **ptrace**( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

EIO	The request code is invalid.
	The given signal number is invalid.
	The specified address is out of bounds.
EPERM	The specified process cannot be traced.
ESRCH	The specified process does not exist.
	request requires process *pid to be* traced by the current process and stopped, and process *pid* is not being traced by the current process.
	request requires process *pid to be* traced by the current process and stopped, and process *pid* is not stopped.

SEE ALSO

adb(1), **intro**(2), **ioctl**(2), **sigvec**(2), **wait**(2V)

BUGS

ptrace( ) is unique and arcane; it should be replaced with a special file which can be opened and read and written. The control functions could then be implemented with **ioctl**(2) calls on this file. This would be simpler to understand and have much higher performance.

The requests **PTRACE_TRACEME** through **PTRACE_SINGLESTEP** are standard UNIX system **ptrace**( ) requests. The requests **PTRACE_ATTACH** through **PTRACE_DUMPCORE** and the fifth argument, *addr2*, are unique to SunOS.

The request **PTRACE_TRACEME** should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use "illegal instruction" signals at a very high rate) could be efficiently debugged.

The error indication, −1, is a legitimate function value; **errno**, (see **intro**(2)), can be used to clarify what it means.

NAME

putmsg – send a message on a stream

SYNOPSIS

#include <stropts.h>

```
int putmsg(fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int flags;
```

DESCRIPTION

putmsg() creates a message (see **intro**(2)) from user specified buffer(s) and sends the message to a STREAMS file. The message may contain either a data part, a control part or both. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The semantics of each part is defined by the STREAMS module that receives the message.

fd specifies a file descriptor referencing an open *stream*. *ctlptr* and *dataptr* each point to a **strbuf** structure that contains the following members:

```
int maxlen;     /* not used */
int len;        /* length of data */
char *buf;      /* ptr to buffer */
```

ctlptr points to the structure describing the control part, if any, to be included in the message. The **buf** field in the **strbuf** structure points to the buffer where the control information resides, and the **len** field indicates the number of bytes to be sent. The **maxlen** field is not used in **putmsg()** (see **getmsg**(2)). In a similar manner, *dataptr* specifies the data, if any, to be included in the message. *flags* may be set to the values 0 or **RS_HIPRI** and is used as described below.

To send the data part of a message, *dataptr* must not be a NULL pointer and the **len** field of *dataptr* must have a value of 0 or greater. To send the control part of a message, the corresponding values must be set for *ctlptr*. No data (control) part will be sent if either *dataptr* (*ctlptr*) is a NULL pointer or the **len** field of *dataptr* (*ctlptr*) is set to –1.

If a control part is specified, and *flags* is set to **RS_HIPRI**, a *priority* message is sent. If *flags* is set to 0, a non-priority message is sent. If no control part is specified, and *flags* is set to **RS_HIPRI**, **putmsg()** fails and sets **errno** to EINVAL. If no control part and no data part are specified, and *flags* is set to 0, no message is sent, and 0 is returned.

For non-priority messages, **putmsg()** will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, **putmsg()** does not block on this condition. For non-priority messages, **putmsg()** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets **errno** to EAGAIN.

putmsg() also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

RETURN VALUES

putmsg() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EAGAIN A non-priority message was specified, the **O_NDELAY** flag is set and the *stream* write queue is full due to internal flow control conditions.

Buffers could not be allocated for the message that was to be created.

EBADF	*fd* is not a valid file descriptor open for writing.
EFAULT	*ctlptr* or *dataptr* points outside the allocated address space.
EINTR	A signal was caught during the **putmsg()** system call.
EINVAL	An undefined value was specified in *flags*.
	flags is set to **RS_HIPRI** and no control part was supplied.
	The *stream* referenced by *fd* is linked below a multiplexor.
ENOSTR	A *stream* is not associated with *fd*.
ENXIO	A hangup condition was generated downstream for the specified *stream*.
ERANGE	The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module.
	The control part of the message is larger than the maximum configured size of the control part of a message.
	The data part of the message is larger than the maximum configured size of the data part of a message.

A **putmsg()** also fails if a STREAMS error message had been processed by the *stream* head before the call to **putmsg()**. The error returned is the value contained in the STREAMS error message.

SEE ALSO

getmsg(2), **intro**(2), **poll**(2), **read**(2V), **write**(2V)

NAME

quotactl – manipulate disk quotas

SYNOPSIS

#include <ufs/quota.h>

```
int quotactl(cmd, special, uid, addr)
int cmd;
char *special;
int uid;
caddr_t addr;
```

DESCRIPTION

The **quotactl**( ) call manipulates disk quotas. *cmd* indicates a command to be applied to the user ID *uid*. *special* is a pointer to a null-terminated string containing the path name of the block special device for the file system being manipulated. The block special device must be mounted as a UFS file system (see **mount**(2V)). *addr* is the address of an optional, command specific, data structure which is copied in or out of the system. The interpretation of *addr* is given with each command below.

Q_QUOTAON Turn on quotas for a file system. *addr* points to the path name of file containing the quotas for the file system. The quota file must exist; it is normally created with the **quotacheck**(8) program. This call is restricted to the super-user.

Q_QUOTAOFF Turn off quotas for a file system. *addr* and *uid* are ignored. This call is restricted to the super-user.

Q_GETQUOTA Get disk quota limits and current usage for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **<ufs/quota.h>**). Only the super-user may get the quotas of a user other than himself.

Q_SETQUOTA Set disk quota limits and current usage for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **quota.h**). This call is restricted to the super-user.

Q_SETQLIM Set disk quota limits for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **quota.h**). This call is restricted to the super-user.

Q_SYNC Update the on-disk copy of quota usages for a file system. If *special* is null then all file systems with active quotas are sync'ed. *addr* and *uid* are ignored.

RETURN VALUES

quotactl( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EFAULT	*addr* or *special* are invalid.
EINVAL	The kernel has not been compiled with the QUOTA option.
	cmd is invalid.
ENODEV	*special* is not a mounted UFS file system.
ENOENT	The file specified by *special* or *addr* does not exist.
ENOTBLK	*special* is not a block device.
EPERM	The call is privileged and the caller was not the super-user.
ESRCH	No disc quota is found for the indicated user.
	Quotas have not been turned on for this file system.
EUSERS	The quota table is full.

If *cmd* is **Q_QUOTAON** **quotactl**() may set errno to:

EACCES — The quota file pointed to by *addr* exists but is not a regular file.

The quota file pointed to by *addr* exists but is not on the file system pointed to by *special*.

EBUSY — **Q_QUOTAON** attempted while another **Q_QUOTAON** or **Q_QUOTAOFF** is in progress.

SEE ALSO

quota(1), **getrlimit**(2), **mount**(2V), **quotacheck**(8), **quotaon**(8)

BUGS

There should be some way to integrate this call with the resource limit interface provided by **setrlimit**( ) and **getrlimit**(2).

Incompatible with Melbourne quotas.

NAME

read, readv – read input

SYNOPSIS

```
int read(fd, buf, nbyte)
int fd;
char *buf;
int nbyte;

#include <sys/types.h>
#include <sys/uio.h>

int readv(fd, iov, iovcnt)
int fd;
struct iovec *iov;
int iovcnt;
```

DESCRIPTION

read() attempts to read *nbyte* bytes of data from the object referenced by the descriptor *fd* into the buffer pointed to by *buf*. **readv()** performs the same action as **read()**, but scatters the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0],*iov*[1], . . . , *iov*[*iovcnt* – 1].

If *nbyte* is zero, **read()** takes no action and returns 0. **readv()**, however, returns –1 and sets the global variable **errno** (see ERRORS below).

For **readv()**, the **iovec** structure is defined as

```
struct iovec {
        caddr_t iov_base;
        int     iov_len;
};
```

Each **iovec** entry specifies the base address and length of an area in memory where data should be placed. **readv()** will always fill an area completely before proceeding to the next.

On objects capable of seeking, the **read()** starts at a position given by the pointer associated with *fd* (see **lseek**(2V)). Upon return from **read()**, the pointer is incremented by the number of bytes actually read.

Objects that are not capable of seeking always read from the current position. The value of the pointer associated with such an object is undefined.

Upon successful completion, **read()** and **readv()** return the number of bytes actually read and placed in the buffer. The system guarantees to read the number of bytes requested if the descriptor references a normal file which has that many bytes left before the EOF (end of file), but in no other case.

If the process calling **read()** or **readv()** receives a signal before any data are read, the system call is restarted unless the process explicitly set the signal to interrupt the call using **sigvec()** or **sigaction()** (see the discussions of **SV_INTERRUPT** on **sigvec**(2) and **SA_INTERRUPT** on **sigaction**(3V)). If **read()** or **readv()** is interrupted by a signal after successfully reading some data, it returns the number of bytes read.

If *nbyte* is not zero and **read()** returns 0, then EOF has been reached. If **readv()** returns 0, then EOF has been reached.

A **read()** or **readv()** from a STREAMS file (see **intro**(2)) can operate in three different modes: "byte-stream" mode, "message-nondiscard" mode, and "message-discard" mode. The default is byte-stream mode. This can be changed using the **I_SRDOPT ioctl**(2) request (see **streamio**(4)), and can be tested with the **I_GRDOPT ioctl()** request. In byte-stream mode, **read()** and **readv()** will retrieve data from the *stream* until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, **read()** and **readv()** will retrieve data until as many bytes as were requested are transferred, or until a message boundary is reached. If the **read()** or **readv()** does not retrieve all the data in a message, the remaining data are left on the *stream*, and can be retrieved by the next **read( )**, **readv( )**, or **getmsg**(2) call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the **read( )** or **readv( )** returns are discarded, and are not available for a subsequent **read( )**, **readv( )**, or **getmsg( )**.

When attempting to read from a descriptor associated with an empty pipe, socket, FIFO, or *stream*:

- If the object the descriptor is associated with is marked for 4.2BSD-style non-blocking I/O (with the **FIONBIO ioctl( )** request or a call to **fcntl**(2V) using the **FNDELAY** flag from **<sys/file.h>** or the **O_NDELAY** flag from **<fcntl.h>** in the 4.2BSD environment), the read will return −1 and **errno** will be set to EWOULDBLOCK.
- If the descriptor is marked for System V-style non-blocking I/O (using **fcntl( )** with the **FNBIO** flag from **<sys/file.h>** or the **O_NDELAY** flag from **<fcntl.h>** in the System V environment), and does not refer to a *stream*, the read will return 0. Note: this is indistinguishable from EOF.
- If the descriptor is marked for POSIX-style non-blocking I/O (using **fcntl( )** with the **O_NONBLOCK** flag from **<fcntl.h>**) and refers to a *stream*, the read will return −1 and **errno** will be set to EAGAIN.
- If neither the descriptor nor the object it refers to are marked for non-blocking I/O, the read will block until data is available to be read or the object has been "disconnected". A pipe or FIFO is "disconnected" when no process has the object open for writing; a socket that was connected is "disconnected" when the connection is broken; a stream is "disconnected" when a hangup condition occurs (for instance, when carrier drops on a terminal).

If the descriptor or the object is marked for non-blocking I/O, and less data are available than are requested by the **read( )** or **readv( )**, only the data that are available are returned, and the count indicates how many bytes of data were actually read.

When reading from a STREAMS file, handling of zero-byte messages is determined by the current read mode setting. In byte-stream mode, **read( )** and **readv( )** accept data until as many bytes as were requested are transferred, or until there is no more data to read, or until a zero-byte message block is encountered. **read( )** and **readv( )** then return the number of bytes read, and places the zero-byte message back on the *stream* to be retrieved by the next **read( )**, **readv( )**, or **getmsg( )**. In the two other modes, a zero-byte message returns a value of 0 and the message is removed from the *stream*. When a zero-byte message is read as the first message on a *stream*, a value of 0 is returned regardless of the read mode.

A **read( )** or **readv( )** from a STREAMS file can only process data messages. It cannot process any type of protocol message and will fail if a protocol message is encountered at the **stream**head.

Upon successful completion, **read( )** and **readv( )** mark for update the **st_atime** field of the file.

RETURN VALUES

read( ) and **readv( )** return the number of bytes actually read on success. On failure, they return −1 and set **errno** to indicate the error.

ERRORS

EAGAIN	The descriptor referred to a *stream*, was marked for System V-style non-blocking I/O, and no data were ready to be read.
EBADF	*d* is not a valid file descriptor open for reading.
EBADMSG	The message waiting to be read on a *stream* is not a data message.
EFAULT	*buf* points outside the allocated address space.
EINTR	The process performing a read from a slow device received a signal before any data arrived, and the signal was set to interrupt the system call.
EINVAL	The *stream* is linked below a multiplexor.
	The pointer associated with *fd* was negative.

EIO　　An I/O error occurred while reading from or writing to the file system.

The calling process is in a background process group and is attempting to read from its controlling terminal and the process is ignoring or blocking SIGTTIN.

The calling process is in a background process group and is attempting to read from its controlling terminal and the process is orphaned.

EISDIR　　*fd* refers to a directory which is on a file system mounted using the NFS.

EWOULDBLOCK　　The file was marked for 4.2BSD-style non-blocking I/O, and no data were ready to be read.

In addition to the above, **readv( )** may set **errno** to:

EFAULT　　Part of *iov* points outside the process's allocated address space.

EINVAL　　*iovcnt* was less than or equal to 0, or greater than 16.

One of the **iov_len** values in the *iov* array was negative.

The sum of the **iov_len** values in the *iov* array overflowed a 32-bit integer.

A **read( )** or **readv( )** from a STREAMS file will also fail if an error message is received at the *stream* head. In this case, **errno** is set to the value returned in the error message. If a hangup occurs on the *stream* being read, **read( )** will continue to operate normally until the **stream head** read queue is empty. Thereafter, it will return 0.

SEE ALSO

dup(2V), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **lseek**(2V), **open**(2V), **pipe**(2V), **select**(2), **socket**(2), **socketpair**(2), **streamio**(4), **termio**(4)

NAME

readlink – read value of a symbolic link

SYNOPSIS

int readlink(path, buf, bufsiz)
char *path, *buf;
int bufsiz;

DESCRIPTION

readlink( ) places the contents of the symbolic link referred to by *path* in the buffer *buf* which has size *bufsiz*. The contents of the link are not null terminated when returned.

RETURN VALUES

readlink( ) returns the number of characters placed in the buffer on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

readlink( ) will fail and the buffer will be unchanged if:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*path* or *buf* extends outside the process's allocated address space.
ELOOP	Too many symbolic links were encountered in translating *path*.
EINVAL	The named file is not a symbolic link.
EIO	An I/O error occurred while reading from or writing to the file system.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The named file does not exist.

SEE ALSO

stat(2V), **symlink**(2)

NAME

reboot – reboot system or halt processor

SYNOPSIS

#include <sys/reboot.h>

reboot(howto, [bootargs])
int howto;
char *bootargs;

DESCRIPTION

reboot() reboots the system, and is invoked automatically in the event of unrecoverable system failures. *howto* is a mask of options passed to the bootstrap program. The system call interface permits only **RB_HALT** or **RB_AUTOBOOT** to be passed to the reboot program; the other flags are used in scripts stored on the console storage media, or used in manual bootstrap procedures. When none of these options (for instance **RB_AUTOBOOT**) is given, the system is rebooted from file **/vmunix** in the root file system of unit 0 of a disk chosen in a processor specific way. An automatic consistency check of the disks is then normally performed.

The bits of *howto* are:

RB_HALT — the processor is simply halted; no reboot takes place. **RB_HALT** should be used with caution.

RB_ASKNAME — Interpreted by the bootstrap program itself, causing it to inquire as to what file should be booted. Normally, the system is booted from the file **/vmunix** without asking.

RB_SINGLE — Normally, the reboot procedure involves an automatic disk consistency check and then multi-user operations. **RB_SINGLE** prevents the consistency check, rather simply booting the system with a single-user shell on the console. **RB_SINGLE** is interpreted by the **init**(8) program in the newly booted system.

RB_DUMP — A system core dump is performed before rebooting.

RB_STRING — The optional argument *bootargs* is passed to the bootstrap program. See **boot**(8S) for details. This option overrides **RB_SINGLE** but the same effect can be achieved by including **–s** as an option in *bootargs*.

Only the super-user may **reboot()** a machine.

RETURN VALUES

On success, **reboot()** does not return. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EPERM — The caller is not the super-user.

FILES

/vmunix

SEE ALSO

panic(8S), **halt**(8), **init**(8), **intro**(8), **reboot**(8)

NAME

recv, recvfrom, recvmsg – receive a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(s, buf, len, flags)
int s;
char *buf;
int len, flags;

int recvfrom(s, buf, len, flags, from, fromlen)
int s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;

int recvmsg(s, msg, flags)
int s;
struct msghdr *msg;
int flags;
```

DESCRIPTION

s is a socket created with **socket**(2). **recv**( ), **recvfrom**( ), and **recvmsg**( ) are used to receive messages from another socket. **recv**( ) may be used only on a *connected* socket (see **connect**(2)), while **recvfrom**( ) and **recvmsg**( ) may be used to receive data on a socket whether it is in a connected state or not.

If *from* is not a NULL pointer, the source address of the message is filled in. *fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see **socket**(2)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking (see **ioctl**(2)) in which case −1 is returned with the external variable **errno** set to EWOULDBLOCK.

The **select**(2) call may be used to determine when more data arrive.

If the process calling **recv**( ), **recvfrom**( ) or **recvmsg**( ) receives a signal before any data are available, the system call is restarted unless the calling process explicitly set the signal to interrupt these calls using **sigvec**( ) or **sigaction**( ) (see the discussions of **SV_INTERRUPT** on **sigvec**(2), and **SA_INTERRUPT** on **sigaction**(3V)).

The *flags* parameter is formed by ORing one or more of the following:

MSG_OOB Read any "out-of-band" data present on the socket, rather than the regular "in-band" data.

MSG_PEEK "Peek" at the data present on the socket; the data are returned, but not consumed, so that a subsequent receive operation will see the same data.

The **recvmsg()** call uses a **msghdr** structure to minimize the number of directly supplied parameters. This structure is defined in **<sys/socket.h>**, and includes the following members:

```
caddr_t       msg_name;         /* optional address */
int           msg_namelen;      /* size of address */
struct iovec  *msg_iov;         /* scatter/gather array */
int           msg_iovlen;       /* # elements in msg_iov */
caddr_t       msg_accrights;    /* access rights sent/received */
int           msg_accrightslen;
```

Here **msg_name** and **msg_namelen** specify the destination address if the socket is unconnected; **msg_name** may be given as a NULL pointer if no names are desired or required. The **msg_iov** and **msg_iovlen** describe the scatter-gather locations, as described in **read**(2V). A buffer to receive any access rights sent along with the message is specified in **msg_accrights**, which has length **msg_accrightslen**.

RETURN VALUES

These calls return the number of bytes received, or −1 if an error occurred.

ERRORS

EBADF — *s* is an invalid descriptor.

EFAULT — The data were specified to be received into a non-existent or protected part of the process address space.

EINTR — The calling process received a signal before any data were available to be received, and the signal was set to interrupt the system call.

ENOTSOCK — *s* is a descriptor for a file, not a socket.

EWOULDBLOCK — The socket is marked non-blocking and the requested operation would block.

SEE ALSO

connect(2), **fcntl**(2V), **getsockopt**(2), **ioctl**(2), **read**(2V), **select**(2), **send**(2), **socket**(2)

NAME

rename – change the name of a file

SYNOPSIS

int rename(path1, path2)
char *path1, *path2;

DESCRIPTION

rename( ) renames the link named *path1* as *path2*. If *path2* exists, then it is first removed. If *path2* refers to a directory, it must be an empty directory, and must not include *path1* in its path prefix. Both *path1* and *path2* must be of the same type (that is, both directories or both non-directories), and must reside on the same file system. Write access permission is required for both the directory containing *path1* and the directory containing *path2*. If a rename request relocates a directory in the hierarchy, write permission in the directory to be moved is needed, since its entry for the parent directory (..) must be updated.

rename( ) guarantees that an instance of *path2* will always exist, even if the system should crash in the middle of the operation.

If the final component of *path1* is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.

If the file referred to by *path2* exists and the file's link count becomes zero when it is removed and no process has the file open, the space occupied by the file is freed, and the file is no longer accessible. If one or more processes have the file open when the last link is removed, the link is removed before **rename( )** returns, but the file's contents are not removed until all references to the file have been closed.

Upon successful completion, **rename( )** marks for update the **st_ctime** and **st_mtime** fields of the parent directory of each file.

RETURN VALUES

rename( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

rename( ) will fail and neither *path1* nor *path2* will be affected if:

EACCES	Write access is denied for either *path1* or *path2*.
	A component of the path prefix of either *path1* or *path2* denies search permission.
	The requested rename requires writing in a directory with access permissions that deny write permission.
EBUSY	*path2* is a directory and is the mount point for a mounted file system.
EDQUOT	The directory in which the entry for the new name is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
EFAULT	Either or both of *path1* or *path2* point outside the process's allocated address space.
EINVAL	*path1* is a parent directory of *path2*.
	An attempt was made to rename '.' or '..'.
EIO	An I/O error occurred while reading from or writing to the file system.
EISDIR	*path2* points to a directory and *path1* points to a file that is not a directory.
ELOOP	Too many symbolic links were encountered while translating either *path1* or *path2*.

ENAMETOOLONG	The length of either path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix of either *path1* or *path2* does not exist.
	The file named by *path1* does not exist.
ENOSPC	The directory in which the entry for the new name is being placed cannot be extended because there is no space left on the file system containing the directory.
ENOTDIR	A component of the path prefix of either *path1* or *path2* is not a directory.
	path1 names a directory and *path2* names a nondirectory file.
ENOTEMPTY	*path2* is a directory and is not empty.
EROFS	The requested rename requires writing in a directory on a read-only file system.
EXDEV	The link named by *path2* and the file named by *path1* are on different logical devices (file systems).

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path1* or *path2* points to an empty string.

SEE ALSO

open(2V)

WARNINGS

The system can deadlock if a loop in the file system graph is present. This loop takes the form of an entry in directory **a**, say **a/file1**, being a hard link to directory **b**, and an entry in directory **b**, say **b/file2**, being a hard link to directory **a**. When such a loop exists and two separate processes attempt to perform '**rename a/file1 b/file2**' and '**rename b/file2 a/file1**', respectively, the system may deadlock attempting to lock both directories for modification. Hard links to directories should not be used. System administrators should use symbolic links instead.

NAME

rmdir – remove a directory file

SYNOPSIS

int rmdir(path)
char *path;

DESCRIPTION

rmdir( ) removes a directory file whose name is given by *path*. The directory must not have any entries other than '.' and '..'. The directory must not be the root directory or the current directory of the calling process.

If the directory's link count becomes zero, and no process has the directory open, the space occupied by the directory is freed and the directory is no longer accessible. If one or more processes have the directory open when the last link is removed, the '.' and '..'. entries, if present, are removed before **rmdir( )** returns and no new entries may be created in the directory, but the directory is not removed until all references to the directory have been closed.

Upon successful completion, **rmdir( )** marks for update the **st_ctime** and **st_mtime** fields of the parent directory.

RETURN VALUES

rmdir( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
EACCES	Write permission is denied for the parent directory of the directory to be removed.
EBUSY	The directory to be removed is the mount point for a mounted file system, or is being used by another process.
EFAULT	*path* points outside the process's allocated address space.
EINVAL	The directory referred to by *path* is the current directory, '.'.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The directory referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
ENOTDIR	The file referred to by *path* is not a directory.
ENOTEMPTY	The directory referred to by *path* contains files other than '.' and '..'.
EROFS	The directory to be removed resides on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to a null pathname.

SEE ALSO

mkdir(2V), **unlink**(2V)

NAME

select – synchronous I/O multiplexing

SYNOPSIS

#include <sys/types.h>
#include <sys/time.h>

int select (width, readfds, writefds, exceptfds, timeout)
int width;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET (fd, &fdset)
FD_CLR (fd, &fdset)
FD_ISSET (fd, &fdset)
FD_ZERO (&fdset)
int fd;
fd_set fdset;

DESCRIPTION

select( ) examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, ready for writing, or have an exceptional condition pending. *width* is the number of bits to be checked in each bit mask that represent a file descriptor; the descriptors from 0 through *width*−1 in the descriptor sets are examined. Typically *width* has the value returned by **ulimit**(3C) for the maximum number of file descriptors. On return, **select( )** replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: **FD_ZERO** (*&fdset*) initializes a descriptor set *fdset* to the null set. **FD_SET**(*fd, &fdset*) includes a particular descriptor *fd* in *fdset*. **FD_CLR**(*fd, &fdset*) removes *fd* from *fdset*. **FD_ISSET**(*fd, &fdset*) is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to **FD_SETSIZE**, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a NULL pointer, the select blocks indefinitely. To effect a poll, the *timeout* argument should be a non-NULL pointer, pointing to a zero-valued **timeval** structure.

Any of *readfds*, *writefds*, and *exceptfds* may be given as NULL pointers if no descriptors are of interest.

Selecting true for reading on a socket descriptor upon which a **listen**(2) call has been performed indicates that a subsequent **accept**(2) call on that descriptor will not block.

RETURN VALUES

select( ) returns a non-negative value on success. A positive value indicates the number of ready descriptors in the descriptor sets. 0 indicates that the time limit referred to by *timeout* expired. On failure, **select( )** returns −1, sets **errno** to indicate the error, and the descriptor sets are not changed.

ERRORS

EBADF	One of the descriptor sets specified an invalid descriptor.
EFAULT	One of the pointers given in the call referred to a non-existent portion of the process' address space.
EINTR	A signal was delivered before any of the selected events occurred, or before the time limit expired.
EINVAL	A component of the pointed-to time limit is outside the acceptable range: **t_sec** must be between 0 and 10^8, inclusive. **t_usec** must be greater than or equal to 0, and less than 10^6.

SEE ALSO

accept(2), **connect**(2), **fcntl**(2V), **ulimit**(3C), **gettimeofday**(2), **listen**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V)

NOTES

Under rare circumstances, **select( )** may indicate that a descriptor is ready for writing when in fact an attempt to write would block. This can happen if system resources necessary for a write are exhausted or otherwise unavailable. If an application deems it critical that writes to a file descriptor not block, it should set the descriptor for non-blocking I/O using the **F_SETFL** request to **fcntl**(2V).

BUGS

Although the provision of **ulimit**(3C) was intended to allow user programs to be written independent of the kernel limit on the number of open files, the dimension of a sufficiently large bit field for select remains a problem. The default size **FD_SETSIZE** (currently 256) is somewhat larger than the current kernel limit to the number of open files. However, in order to accommodate programs which might potentially use a larger number of open files with select, it is possible to increase this size within a program by providing a larger definition of **FD_SETSIZE** before the inclusion of **<sys/types.h>**.

select( ) should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout pointer will be unmodified by the **select( )** call.

NAME

semctl – semaphore control operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(semid, semnum, cmd, arg)
int semid, semnum, cmd;
union semun {
        val;
        struct semid_ds *buf;
        ushort *array;
} arg;
```

DESCRIPTION

semctl( ) provides a variety of semaphore control operations as specified by *cmd*.

The following *cmd*s are executed with respect to the semaphore specified by *semid* and *semnum:*

GETVAL Return the value of *semval* (see **intro**(2)). **[READ]**

SETVAL Set the value of *semval* to *arg.val*. **[ALTER]** When this cmd is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared.

GETPID Return the value of *sempid*. **[READ]**

GETNCNT Return the value of *semncnt*. **[READ]**

GETZCNT Return the value of *semzcnt*. **[READ]**

The following *cmd*s return and set, respectively, every *semval* in the set of semaphores.

GETALL Place *semvals* into the array pointed to by *arg.array*. **[READ]**

SETALL Set *semvals* according to the array pointed to by *arg.array*. **[ALTER]** When this cmd is successfully executed the *semadj* values corresponding to each specified semaphore in all processes are cleared.

The following *cmd*s are also available:

IPC_STAT Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in **intro**(2). **[READ]**

IPC_SET Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*:

sem_perm.uid
sem_perm.gid
sem_perm.mode /* only low 9 bits */

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*.

IPC_RMID Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*.

In the **semop**(2) and **semctl**(2) system call descriptions, the permission required for an operation is given as "[token]", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Alter by user
00060	Read, Alter by group
00006	Read, Alter by others

Read and Alter permissions on a semid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **sem_perm.[c]uid** in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process matches **sem_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process does not match **sem_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

RETURN VALUES

On success, the value returned by **semctl**( ) depends on *cmd* as follows:

GETVAL	The value of *semval*.
GETPID	The value of *sempid*.
GETNCNT	The value of *semncnt*.
GETZCNT	The value of *semzcnt*.
All others	0.

On failure, **semctl**( ) returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES	Operation permission is denied to the calling process (see **intro**(2)).
EFAULT	*arg.buf* points to an illegal address.
EINVAL	*semid* is not a valid semaphore identifier.
	semnum is less than zero or greater than **sem_nsems**.
	cmd is not a valid command.
EPERM	*cmd* is **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not super-user.
	cmd is **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*.
ERANGE	*cmd* is **SETVAL** or **SETALL** and the value to which *semval* is to be set is greater than the system imposed maximum.

SEE ALSO

intro(2), **semget**(2), **semop**(2), **ipcrm**(1), **ipcs**(1)

NAME

semget – get set of semaphores

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key, nsems, semflg)
key_t key;
int nsems, semflg;
```

DESCRIPTION

semget() returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see **intro**(2)) are created for *key* if one of the following are true:

- *key* is equal to **IPC_PRIVATE**.
- *key* does not already have a semaphore identifier associated with it, and (*semflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

- **sem_perm.cuid**, **sem_perm.uid**, **sem_perm.cgid**, and **sem_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **sem_perm.mode** are set equal to the low-order 9 bits of *semflg*.
- **sem_nsems** is set equal to the value of *nsems*.
- **sem_otime** is set equal to 0 and **sem_ctime** is set equal to the current time.

A semaphore identifier (semid) is a unique positive integer created by a **semget**(2) system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as **semid_ds** and contains the following members:

```
struct   ipc_perm sem_perm;   /* operation permission struct */
ushort   sem_nsems;           /* number of sems in set */
time_t   sem_otime;           /* last operation time */
time_t   sem_ctime;           /* last change time */
                              /* Times measured in secs since */
                              /* 00:00:00 GMT, Jan. 1, 1970 */
```

sem_perm is an **ipc_perm** structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort   cuid;                /* creator user id */
ushort   cgid;                /* creator group id */
ushort   uid;                 /* user id */
ushort   gid;                 /* group id */
ushort   mode;                /* r/a permission */
```

The value of **sem_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a **sem_num**. **sem_num** values run sequentially from 0 to the value of **sem_nsems** minus 1. **sem_otime** is the time of the last **semop**(2) operation, and **sem_ctime** is the time of the last **semctl**(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort  semval;     /* semaphore value */
short   sempid;     /* pid of last operation */
ushort  semncnt;    /* # awaiting semval > cval */
ushort  semzcnt;    /* # awaiting semval = 0 */
```

semval is a non-negative integer. **sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become greater than its current value. **semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become zero.

RETURN VALUES

semget() returns a non-negative semaphore identifier on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EACCES — A semaphore identifier exists for *key*, but operation permission (see **intro**(2)) as specified by the low-order 9 bits of *semflg* would not be granted.

EEXIST — A semaphore identifier exists for *key* but ((*semflg* & **IPC_CREAT**) and (*semflg* & **IPC_EXCL**)) is "true".

EINVAL — *nsems* is either less than or equal to zero or greater than the system-imposed limit.

A semaphore identifier exists for *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to zero.

ENOENT — A semaphore identifier does not exist for *key* and (*semflg* & **IPC_CREAT**) is "false".

ENOSPC — A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded.

A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded.

SEE ALSO

ipcrm(1), **ipcs**(1), **intro**(2), **semctl**(2), **semop**(2)

NAME

semop – semaphore operations

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(semid, sops, nsops)
int semid;
struct sembuf *sops;
int nsops;

DESCRIPTION

semop() is used to perform atomically an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *sops* is a pointer to the array of semaphore-operation structures. *nsops* is the number of such structures in the array. The contents of each structure includes the following members:

```
short   sem_num;   /* semaphore number */
short   sem_op;    /* semaphore operation */
short   sem_flg;   /* operation flags */
```

Each semaphore operation specified by **sem_op** is performed on the corresponding semaphore specified by *semid* and *sem_num*.

sem_op specifies one of three semaphore operations as follows:

If **sem_op** is a negative integer, one of the following will occur: **[ALTER]** (see **semctl**(2))

- If *semval* (see **intro**(2)) is greater than or equal to the absolute value of **sem_op()**, the absolute value of **sem_op()** is subtracted from *semval*. Also, if (**sem_flg** & **SEM_UNDO**) is "true", the absolute value of **sem_op()** is added to the calling process's *semadj* value (see **exit**(2V)) for the specified semaphore.
- If *semval* is less than the absolute value of **sem_op()** and (**sem_flg** & **IPC_NOWAIT**) is "true", **semop()** will return immediately.
- If *semval* is less than the absolute value of **sem_op()** and (**sem_flg** & **IPC_NOWAIT**) is "false", **semop()** will increment the *semncnt* associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occur.

  semval becomes greater than or equal to the absolute value of **sem_op()**. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, the absolute value of **sem_op()** is subtracted from *semval* and, if (**sem_flg** & **SEM_UNDO**) is "true", the absolute value of **sem_op()** is added to the calling process's *semadj* value for the specified semaphore.

  The *semid* for which the calling process is awaiting action is removed from the system (see **semctl**(2)). When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.

  The calling process receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in **signal**(3V).

If **sem_op()** is a positive integer, the value of **sem_op()** is added to *semval* and, if (**sem_flg** & **SEM_UNDO**) is "true", the value of **sem_op()** is subtracted from the calling process's *semadj* value for the specified semaphore. **[ALTER]**

If **sem_op()** is zero, one of the following will occur: **[READ]**

- If *semval* is zero, **semop()** will return immediately.
- If *semval* is not equal to zero and **(sem_flg & IPC_NOWAIT)** is "true", **semop()** will return immediately.
- If *semval* is not equal to zero and **(sem_flg & IPC_NOWAIT)** is "false", **semop()** will increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:
    - *semval* becomes zero, at which time the value of *semzcnt* associated with the specified semaphore is decremented.
    - The *semid* for which the calling process is awaiting action is removed from the system. When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.
    - The calling process receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in **signal**(3V).

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

RETURN VALUES

semop() returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

E2BIG	*nsops* is greater than the system-imposed maximum.
EACCES	Operation permission is denied to the calling process (see **intro**(2)).
EAGAIN	The operation would result in suspension of the calling process but **(sem_flg & IPC_NOWAIT)** is "true".
EFAULT	*sops* points to an illegal address.
EFBIG	**sem_num** is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*.
EIDRM	The set of semaphores referred to by *msqid* was removed from the system.
EINTR	The call was interrupted by the delivery of a signal.
EINVAL	*semid* is not a valid semaphore identifier.
	The number of individual semaphores for which the calling process requests a **SEM_UNDO** would exceed the limit.
ENOSPC	The limit on the number of individual processes requesting an **SEM_UNDO** would be exceeded.
ERANGE	An operation would cause a *semval* or *semadj* value to overflow the system-imposed limit.

SEE ALSO

ipcrm(1), **ipcs**(1), **intro**(2), **execve**(2V), **exit**(2V), **fork**(2V), **semctl**(2), **semget**(2), **signal**(3V)

NAME

send, sendto, sendmsg – send a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int send(s, msg, len, flags)
int s;
char *msg;
int len, flags;

int sendto(s, msg, len, flags, to, tolen)
int s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;

int sendmsg(s, msg, flags)
int s;
struct msghdr *msg;
int flags;
```

DESCRIPTION

s is a socket created with **socket**(2). **send**( ), **sendto**( ), and **sendmsg**( ) are used to transmit a message to another socket. **send**( ) may be used only when the socket is in a *connected* state, while **sendto**( ) and **sendmsg**( ) may be used at any time.

The address of the target is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a **send**( ). Return values of –1 indicate some locally detected errors.

If no buffer space is available at the socket to hold the message to be transmitted, then **send**( ) normally blocks, unless the socket has been placed in non-blocking I/O mode. The **select**(2) call may be used to determine when it is possible to send more data.

If the process calling **send**( ), **sendmsg**( ) or **sendto**( ) receives a signal before any data are buffered to be sent, the system call is restarted unless the calling process explicitly set the signal to interrupt these calls using **sigvec**( ) or **sigaction**( ) (see the discussions of **SV_INTERRUPT** on **sigvec**(2), and **SA_INTERRUPT** on **sigaction**(3V)).

The *flags* parameter is formed by ORing one or more of the following:

MSG_OOB Send "out-of-band" data on sockets that support this notion. The underlying protocol must also support "out-of-band" data. Currently, only **SOCK_STREAM** sockets created in the **AF_INET** address family support out-of-band data.

MSG_DONTROUTE The **SO_DONTROUTE** option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.

See **recv**(2) for a description of the **msghdr** structure.

RETURN VALUES

On success, these functions return the number of bytes sent. On failure, they return –1 and set **errno** to indicate the error.

ERRORS

EBADF	*s* is an invalid descriptor.
EFAULT	The data was specified to be sent to a non-existent or protected part of the process address space.
EINTR	The calling process received a signal before any data could be buffered to be sent, and the signal was set to interrupt the system call.
EINVAL	*len* is not the size of a valid address for the specified address family.
EMSGSIZE	The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.
ENOBUFS	The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.
ENOBUFS	The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion.
ENOTSOCK	*s* is a descriptor for a file, not a socket.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.

SEE ALSO

connect(2), **fcntl**(2V), **getsockopt**(2), **recv**(2), **select**(2), **socket**(2), **write**(2V)

NAME

setpgid – set process group ID for job control

SYNOPSIS

#include <sys/types.h>

int setpgid (pid, pgid)
pid_t pid, pgid;

DESCRIPTION

setpgid( ) is used to either join an existing process group or create a new process group within the session of the calling process (see NOTES). The process group ID of a session leader does not change. Upon successful completion, the process group ID of the process with a process ID that matches *pid* is set to *pgid*. As a special case, if *pid* is zero, the process ID of the calling process is used. Also, if *pgid* is zero, the process ID of the process indicated by *pid* is used.

RETURN VALUES

setpgid( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES The value of *pid* matches the process ID of a child process of the calling process and the child process has successfully executed one of the **exec( )** functions.

EINVAL The value of *pgid* is less than zero or is greater than MAXPID, the maximum process ID as defined in **<sys/param.h>**.

EPERM The process indicated by *pid* is a session leader. The value of *pid* is valid but matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process. The value of *pgid* does not match the process ID of the process indicated by *pid* and there is no process with a process group ID that matches the value of *pgid* in the same session as the calling process.

ESRCH *pid* does not match the PID of the calling process or the PID of a child of the calling process.

SEE ALSO

getpgrp(2V), **execve**(2V), **setsid**(2V), **tcgetpgrp**(3V)

NOTES

For **setpgid**() to behave as described above, {_POSIX_JOB_CONTROL} must be in effect (see **sysconf**(2V)). {_POSIX_JOB_CONTROL} is always in effect on SunOS systems, but for portability, applications should call **sysconf( )** to determine whether {_POSIX_JOB_CONTROL} is in effect for the current system.

NAME

setregid – set real and effective group IDs

SYNOPSIS

int setregid(rgid, egid)
int rgid, egid;

DESCRIPTION

setregid() is used to set the real and effective group IDs of the calling process. If *rgid* is −1, the real GID is not changed; if *egid* is −1, the effective GID is not changed. The real and effective GIDs may be set to different values in the same call.

If the effective user ID of the calling process is super-user, the real GID and the effective GID can be set to any legal value.

If the effective user ID of the calling process is not super-user, either the real GID can be set to the saved setGID from **execve**(2V), or the effective GID can either be set to the saved setGID or the real GID. Note: if a setGID process sets its effective GID to its real GID, it can still set its effective GID back to the saved setGID.

In either case, if the real GID is being changed (that is, if *rgid* is not −1), or the effective GID is being changed to a value not equal to the real GID, the saved setGID is set equal to the new effective GID.

RETURN VALUES

setregid() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

setregid() will fail and neither of the group IDs will be changed if:

EINVAL The value of *rgid* or *egid* is less than 0 or greater than **USHRT_MAX** (defined in **<sys/limits.h>**).

EPERM The calling process' effective UID is not the super-user and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.

SEE ALSO

execve(2V), **getgid**(2V), **setreuid**(2), **setuid**(3V)

NAME

setreuid – set real and effective user IDs

SYNOPSIS

int setreuid(ruid, euid)
int ruid, euid;

DESCRIPTION

setreuid() is used to set the real and effective user IDs of the calling process. If *ruid* is −1, the real user ID is not changed; if *euid* is −1, the effective user ID is not changed. The real and effective user IDs may be set to different values in the same call.

If the effective user ID of the calling process is super-user, the real user ID and the effective user ID can be set to any legal value.

If the effective user ID of the calling process is not super-user, either the real user ID can be set to the effective user ID, or the effective user ID can either be set to the saved set-user ID from **execve**(2V) or the real user ID. Note: if a set-UID process sets its effective user ID to its real user ID, it can still set its effective user ID back to the saved set-user ID.

In either case, if the real user ID is being changed (that is, if *ruid* is not −1), or the effective user ID is being changed to a value not equal to the real user ID, the saved set-user ID is set equal to the new effective user ID.

RETURN VALUES

setreuid() returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

setreuid() will fail and neither of the user IDs will be changed if:

EINVAL	The value of *ruid* or *euid* is less than 0 or greater than **USHRT_MAX** (defined in **<sys/limits.h>**).
EPERM	The calling process' effective user ID is not the super-user and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified.

SEE ALSO

execve(2V), **getuid**(2V), **setregid**(2), **setuid**(3V)

NAME

setsid – create session and set process group ID

SYNOPSIS

#include <sys/types.h>

pid_t setsid( )

DESCRIPTION

If the calling process is not a process group leader, the **setsid( )** function creates a new session. The calling process is the session leader of this new session, the process group leader of a new process group, and has no controlling terminal. If the process had a controlling terminal, **setsid( )** breaks the association between the process and that controlling terminal. The process group ID of the calling process is set equal to the process ID of the calling process. The calling process is the only process in the new process group and the only process in the new session.

RETURN VALUES

setsid( ) returns the process group ID of the calling process on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

If any of the following conditions occur, **setsid( )** returns −1 and sets **errno** to the corresponding value:

EPERM The calling process is already a process group leader.

The process ID of the calling process equals the process group ID of a different process.

SEE ALSO

execve(2V), **exit**(2V), **fork**(2V), **getpid**(2V), **getpgrp**(2V), **kill**(2V), **setpgid**(2V), **sigaction**(3V)

NAME

setuseraudit, setaudit – set the audit classes for a specified user ID

SYNOPSIS

#include <sys/label.h>
#include <sys/audit.h>

int setuseraudit(uid, state)
int uid;
audit_state_t *state;

int setaudit(state)
audit_state_t *state;

DESCRIPTION

The **setuseraudit( )** system call sets the audit state for all processes whose audit user ID matches the specified user ID. The parameter *state* specifies the audit classes to audit for both successful and unsuccessful operations.

The **setaudit( )** system call sets the audit state for the current process.

Only processes with the real or effective user ID of the super-user may successfully execute these calls.

RETURN VALUES

setuseraudit( ) and **setaudit( )** return:

0 on success.

–1 on failure and set **errno** to indicate the error.

ERRORS

EFAULT The *state* parameter points outside the processes' allocated address space.

EPERM The process' real or effective user ID is not super-user.

SEE ALSO

audit(2), **audit_args**(3), **audit_control**(5), **audit.log**(5)

NAME

shmctl – shared memory control operations

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;

DESCRIPTION

shmctl() provides a variety of shared memory control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in **intro**(2). **[READ]**

IPC_SET Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */

This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

IPC_RMID Remove the shared memory identifier specified by *shmid* from the system. If no processes are currently mapped to the corresponding shared memory segment, then the segment is removed and the associated resources are reclaimed. Otherwise, the segment will persist, although **shmget**(2) will not be able to locate it, until it is no longer mapped by any process. This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

In the **shmop**(2) and **shmctl**(2) system call descriptions, the permission required for an operation is given as "[token]", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a shmid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **shm_perm.[c]uid** in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process matches **shm_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process does not match **shm_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

RETURN VALUES

shmctl() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES *cmd* is equal to **IPC_STAT** and **[READ]** operation permission is denied to the calling process (see **intro**(2)).

EFAULT *buf* points to an illegal address.

EINVAL *shmid* is not a valid shared memory identifier.

cmd is not a valid command.

EPERM *cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not super-user or the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

SEE ALSO

ipcrm(1), **ipcs**(1), **intro**(2), **shmget**(2), **shmop**(2)

NAME

shmget – get shared memory segment identifier

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key, size, shmflg)
key_t key;
int size, shmflg;
```

DESCRIPTION

shmget() returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of at least *size* bytes (see **intro**(2)) are created for *key* if one of the following are true:

- *key* is equal to **IPC_PRIVATE**.
- *key* does not already have a shared memory identifier associated with it, and (*shmflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

- **shm_perm.cuid**, **shm_perm.uid**, **shm_perm.cgid**, and **shm_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **shm_perm.mode** are set equal to the low-order 9 bits of *shmflg*.
- **shm_segsz** is set equal to the value of *size*.
- **shm_lpid**, **shm_nattch**, **shm_atime**, and **shm_dtime** are set equal to 0.
- **shm_ctime** is set equal to the current time.

A shared memory identifier (shmid) is a unique positive integer created by a **shmget**(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as **shmid_ds** and contains the following members:

```
struct   ipc_perm shm_perm;  /* operation permission struct */
int      shm_segsz;          /* size of segment */
ushort   shm_cpid;           /* creator pid */
ushort   shm_lpid;           /* pid of last operation */
short    shm_nattch;         /* number of current attaches */
time_t   shm_atime;          /* last attach time */
time_t   shm_dtime;          /* last detach time */
time_t   shm_ctime;          /* last change time */
                             /* Times measured in secs since */
                             /* 00:00:00 GMT, Jan. 1, 1970 */
```

shm_perm is an **ipc_perm** structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort   cuid;               /* creator user id */
ushort   cgid;               /* creator group id */
ushort   uid;                /* user id */
ushort   gid;                /* group id */
ushort   mode;               /* r/w permission */
```

shm_segsz specifies the size of the shared memory segment. **shm_cpid** is the process ID of the process that created the shared memory identifier. **shm_lpid** is the process ID of the last process that performed a **shmop**(2) operation. **shm_nattch** is the number of processes that currently have this segment attached. **shm_atime** is the time of the last *shmat* operation, **shm_dtime** is the time of the last *shmdt* operation, and **shm_ctime** is the time of the last **shmctl**(2) operation that changed one of the members of the above structure.

RETURN VALUES

shmget() returns a non-negative shared memory identifier on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES	A shared memory identifier exists for *key* but operation permission (see **intro**(2)) as specified by the low-order 9 bits of *shmflg* would not be granted.
EEXIST	A shared memory identifier exists for *key* but ((*shmflg* & **IPC_CREAT**) && (*shmflg* & **IPC_EXCL**)) is "true".
EINVAL	*size* is less than the system-imposed minimum or greater than the system-imposed maximum.
	A shared memory identifier exists for *key* but the size of the segment associated with it is less than *size* and *size* is not equal to zero.
ENOENT	A shared memory identifier does not exist for *key* and (*shmflg* & **IPC_CREAT**) is "false".
ENOMEM	A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request.
ENOSPC	A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded.

SEE ALSO

ipcrm(1), **ipcs**(1), **intro**(2), **shmctl**(2), **shmop**(2)

NAME

shmop, shmat, shmdt – shared memory operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat(shmid, shmaddr, shmflg)
int shmid;
char *shmaddr;
int shmflg;

int shmdt(shmaddr)
char *shmaddr;
```

DESCRIPTION

shmat( ) maps the shared memory segment associated with the shared memory identifier specified by *shmid* into the data segment of the calling process. Upon successful completion, the address of the mapped segment is returned.

The shared memory segment is mapped at the address specified by one of the following criteria:

- If *shmaddr* is equal to zero, the segment is mapped at an address selected by the system. Ordinarily, applications should invoke **shmat( )** with *shmaddr* equal to zero so that the operating system may make the best use of available resources.
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is ''true'', the segment is mapped at the address given by (*shmaddr* - (*shmaddr* modulus **SHMLBA**)).
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is ''false'', the segment is mapped at the address given by *shmaddr*.

The segment is mapped for reading if (*shmflg* & **SHM_RDONLY**) is ''true'' **[READ]**, otherwise it is mapped for reading and writing **[READ/WRITE]** (see **shmctl**(2)).

shmdt( ) unmaps from the calling process's address space the shared memory segment that is mapped at the address specified by *shmaddr*. The shared memory segment must have been mapped with a prior **shmat( )** function call. The segment and contents are retained until explicitly removed by means of the **IPC_RMID** function (see **shmctl**(2)).

RETURN VALUES

shmat( ) returns the data segment start address of the mapped shared memory segment. On failure, it returns –1 and sets **errno** to indicate the error.

shmdt( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

shmat( ) will fail and not map the shared memory segment if one or more of the following are true:

EACCES Operation permission is denied to the calling process (see **intro**(2)).

EINVAL *shmid* is not a valid shared memory identifier.

shmaddr is not equal to zero, and the value of (*shmaddr* - (*shmaddr* modulus **SHMLBA**)) is an illegal address.

shmaddr is not equal to zero, (*shmflg* & **SHM_RND**) is ''false'', and the value of *shmaddr* is an illegal address.

EMFILE The number of shared memory segments mapped to the calling process would exceed the system-imposed limit.

ENOMEM The available data space is not large enough to accommodate the shared memory segment.

shmdt( ) will fail and not unmap the shared memory segment if:

EINVAL *shmaddr* is not the data segment start address of a shared memory segment.

SEE ALSO

ipcrm(1), **ipcs**(1), **intro**(2), **execve**(2V), **exit**(2V), **fork**(2V), **shmctl**(2), **shmget**(2)

NAME

shutdown – shut down part of a full-duplex connection

SYNOPSIS

int shutdown(s, how)
int s, how;

DESCRIPTION

The **shutdown()** call causes all or part of a full-duplex connection on the socket associated with *s* to be shut down. If *how* is 0, then further receives will be disallowed. If *how* is 1, then further sends will be disallowed. If *how* is 2, then further sends and receives will be disallowed.

RETURN VALUES

shutdown() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EBADF	*s* is not a valid descriptor.
ENOTCONN	The specified socket is not connected.
ENOTSOCK	*s* is a file, not a socket.

SEE ALSO

ipcrm(1), **ipcs**(1), **connect**(2), **socket**(2)

BUGS

The *how* values should be defined constants.

NAME

sigblock, sigmask – block signals

SYNOPSIS

#include <signal.h>

int sigblock(mask);
int mask;

int sigmask(signum)

DESCRIPTION

sigblock( ) adds the signals specified in *mask* to the set of signals currently being blocked from delivery. A signal is blocked if the appropriate bit in *mask* is set. The macro **sigmask( )** is provided to construct the signal mask for a given *signum*. **sigblock( )** returns the previous signal mask, which may be restored using **sigsetmask**(2).

It is not possible to block **SIGKILL** or **SIGSTOP**. The system silently imposes this restriction.

RETURN VALUES

sigblock( ) returns the previous signal mask.

The **sigmask( )** macro returns the mask for the given signal number.

SEE ALSO

kill(2V), **sigsetmask**(2), **sigvec**(2), **signal**(3V)

NAME

sigpause, sigsuspend – automatically release blocked signals and wait for interrupt

SYNOPSIS

int sigpause(sigmask)
int sigmask;

#include <signal.h>

int sigsuspend(sigmaskp)
sigset_t *sigmaskp;

DESCRIPTION

sigpause() assigns *sigmask* to the set of masked signals and then waits for a signal to arrive; on return the set of masked signals is restored. *sigmask* is usually 0 to indicate that no signals are now to be blocked. **sigpause()** always terminates by being interrupted, returning EINTR.

In normal usage, a signal is blocked using **sigblock**(2), to begin a critical section, variables modified on the occurrence of the signal are examined to determine that there is no work to be done, and the process pauses awaiting work by using **sigpause()** with the mask returned by **sigblock()**.

sigsuspend() replaces the process's signal mask with the set of signals pointed to by *sigmaskp* and then suspends the process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. If the action is to terminate the process, **sigsuspend()** does not return. If the action is to execute a signal-catching function, **sigsuspend()** returns after the signal-catching function returns, with the signal mask restored to the setting that existed prior to the **sigsuspend()** call. It is not possible to block those signals that cannot be ignored, as documented in **<signal.h>** this is enforced by the system without indicating an error.

RETURN VALUES

Since **sigpause()** and **sigsuspend()** suspend process execution indefinitely, there is no successful completion return value. On failure, these functions return −1 and set **errno** to indicate the error.

ERRORS

EINTR A signal is caught by the calling process and control is returned from the signal-catching function.

SEE ALSO

sigblock(2), **sigpending**(2V), **sigprocmask**(2V), **sigvec**(2), **pause**(3V), **sigaction**(3V), **signal**(3V), **sigsetops**(3V)

NAME

sigpending – examine pending signals

SYNOPSIS

#include <signal.h>

int sigpending(set)
sigset_t *set;

DESCRIPTION

sigpending( ) stores the set of signals that are blocked from delivery and pending for the calling process in the space pointed to by *set*.

RETURN VALUES

sigpending( ) returns:

0	on success.

–1	on failure and sets **errno** to indicate the error.

SEE ALSO

sigprocmask(2V), **sigvec**(2), **sigsetops**(3V)

NAME

sigprocmask – examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int sigprocmask(how, set, oset)
int how;
sigset_t *set, *oset;
```

DESCRIPTION

sigprocmask( ) is used to examine or change (or both) the calling process's signal mask. If the value of *set* is not NULL, it points to a set of signals to be used to change the currently blocked set.

The value of *how* indicates the manner in which the set is changed, and consists of one of the following values, as defined in the header **<signal.h>**:

SIG_BLOCK The resulting set is the union of the current set and the signal set pointed to by *set*.

SIG_UNBLOCK The resulting set is the intersection of the current set and the complement of the signal set pointed to by *set*.

SIG_SETMASK The resulting set is the signal set pointed to by *set*.

If *oset* is not NULL, the previous mask is stored in the space pointed to by *oset*. If the value of *set* is NULL, the value of *how* is not significant and the process's signal mask is unchanged by this function call. Thus, the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to **sigprocmask**( ), at least one of those signals is be delivered before **sigprocmask**( ) returns.

If it is not possible to block the **SIGKILL** and **SIGSTOP** signals. This is enforced by the system without causing an error to be indicated.

If any of the **SIGFPE**, **SIGKILL**, or **SIGSEGV** signals are generated while they are blocked, the result is undefined, unless the signal was generated by a call to **kill**(2V).

If **sigprocmask**( ) fails, the process's signal mask is not changed.

RETURN VALUES

sigprocmask( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL The value of *how* is not equal to one of the defined values.

SEE ALSO

sigpause(2V), **sigpending**(2V), **sigvec**(2), **sigaction**(3V), **sigsetops**(3V)

NAME

sigsetmask – set current signal mask

SYNOPSIS

#include <signal.h>

int sigsetmask(mask)
int mask;

DESCRIPTION

sigsetmask() sets the set of signals currently being blocked from delivery according to *mask*. A signal is blocked if the appropriate bit in *mask* is set. The macro **sigblock**(2) is provided to construct the mask for a given *signum*.

The system silently disallows blocking **SIGKILL** and **SIGSTOP**.

RETURN VALUES

sigsetmask() returns the previous signal mask.

SEE ALSO

kill(2V), **sigblock**(2), **sigpause**(2V), **sigvec**(2), **signal**(3V)

NAME

sigstack – set and/or get signal stack context

SYNOPSIS

#include <signal.h>

int sigstack (ss, oss)
struct sigstack *ss, *oss;

DESCRIPTION

sigstack() allows users to define an alternate stack, called the "signal stack", on which signals are to be processed. When a signal's action indicates its handler should execute on the signal stack (specified with a **sigvec**(2) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler's execution.

A signal stack is specified by a **sigstack()** structure, which includes the following members:

```
char        *ss_sp;         /* signal stack pointer */
int         ss_onstack;     /* current status */
```

ss_sp is the initial value to be assigned to the stack pointer when the system switches the process to the signal stack. Note that, on machines where the stack grows downwards in memory, this is *not* the address of the beginning of the signal stack area. **ss_onstack** field is zero or non-zero depending on whether the process is currently executing on the signal stack or not.

If *ss* is not a NULL pointer, **sigstack()** sets the signal stack state to the value in the **sigstack()** structure pointed to by *ss*. Note: if **ss_onstack** is non-zero, the system will think that the process is executing on the signal stack. If *ss* is a NULL pointer, the signal stack state will be unchanged. If *oss* is not a NULL pointer, the current signal stack state is stored in the **sigstack()** structure pointed to by *oss*.

RETURN VALUES

sigstack() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

sigstack() will fail and the signal stack context will remain unchanged if one of the following occurs.

EFAULT *ss* or *oss* points to memory that is not a valid part of the process address space.

SEE ALSO

sigvec(2), **setjmp**(3V), **signal**(3V)

NOTES

Signal stacks are not "grown" automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

NAME

sigvec – software signal facilities

SYNOPSIS

#include <signal.h>

int sigvec(sig, vec, ovec)
int sig;
struct sigvec *vec, *ovec;

DESCRIPTION

The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored*. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special *signal stack*.

All signals have the same *priority*. Signal routines execute with the signal that caused their invocation *blocked*, but other signals may yet occur. A global *signal mask* defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a **sigblock**(2) or **sigsetmask**(2) call, or when a signal is delivered to the process.

A process may also specify a set of *flags* for a signal that affect the delivery of that signal.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a **sigblock**( ) or **sigsetmask**( ) call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and ORing in the signal mask associated with the handler to be invoked.

The action to be taken when the signal is delivered is specified by a **sigvec** structure, defined in **<signal.h>** as:

```
struct sigvec {
        void (*sv_handler)();      /* signal handler */
        int sv_mask;               /* signal mask to apply */
        int sv_flags;              /* see signal options */
}
```

The following bits may be set in **sv_flags**:

```
#define SV_ONSTACK      0x0001   /* take signal on signal stack */
#define SV_INTERRUPT    0x0002   /* do not restart system on signal return */
#define SV_RESETHAND    0x0004   /* reset signal handler to SIG_DFL on signal */
```

If the **SV_ONSTACK** bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with **sigstack**(2), rather than delivering the signal on the current stack.

If *vec* is not a NULL pointer, **sigvec**( ) assigns the handler specified by **sv_handler**, the mask specified by **sv_mask**, and the flags specified by **sv_flags** to the specified signal. If *vec* is a NULL pointer, **sigvec**( ) does not change the handler, mask, or flags for the specified signal.

The mask specified in *vec* is not allowed to block SIGKILL or **SIGSTOP**. The system enforces this restriction silently.

If *ovec* is not a NULL pointer, the handler, mask, and flags in effect for the signal before the call to **sigvec()** are returned to the user. A call to **sigvec()** with *vec* a NULL pointer and *ovec* not a NULL pointer can be used to determine the handling information currently in effect for a signal without changing that information.

The following is a list of all signals with names as in the include file **<signal.h>**:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGABRT	6*	abort (generated by **abort**(3) routine)
SIGEMT	7*	emulator trap
SIGFPE	8*	arithmetic exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe or other socket with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16•	urgent condition present on socket
SIGSTOP	17†	stop (cannot be caught, blocked, or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19•	continue after stop
SIGCHLD	20•	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGIO	23•	I/O is possible on a descriptor (see **fcntl**(2V))
SIGXCPU	24	cpu time limit exceeded (see **getrlimit**(2))
SIGXFSZ	25	file size limit exceeded (see **getrlimit**(2))
SIGVTALRM	26	virtual time alarm (see **getitimer**(2))
SIGPROF	27	profiling timer alarm (see **getitimer**(2))
SIGWINCH	28•	window changed (see **termio**(4) and **win**(4S))
SIGLOST	29*	resource lost (see **lockd**(8C))
SIGUSR1	30	user-defined signal 1
SIGUSR2	31	user-defined signal 2

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another **sigvec()** call is made, or an **execve**(2V) is performed, unless the **SV_RESETHAND** bit is set in the flags for that signal. In that case, the value of the handler for the caught signal is set to **SIG_DFL** before entering the signal-catching function, unless the signal is **SIGILL** or **SIGTRAP**. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The **SV_RESETHAND** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal's handler to **SIG_DFL**; this default is termination except for signals marked with • or †. Signals marked with • are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If the process is terminated, a "core image" will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list *and* the following conditions are met:

- The effective user ID (EUID) and the real user ID (UID) of the receiving process are equal.
- The effective group ID (EGID) and the real group ID (GID) of the receiving process are equal.

- An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:
  - a mode of 0666 modified by the file creation mask (see **umask**(2V))
  - a file owner ID that is the same as the effective user ID of the receiving process.
  - a file group ID that is the same as the file group ID of the current directory

If the handler for that signal is **SIG_IGN**, the signal is subsequently ignored, and pending instances of the signal are discarded.

Note: the signals **SIGKILL** and **SIGSTOP** cannot be ignored.

If a caught signal occurs during certain system calls, the call is restarted by default. The call can be forced to terminate prematurely with an EINTR error return by setting the **SV_INTERRUPT** bit in the flags for that signal. **SV_INTERRUPT** is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected system calls are **read**(2V) or **write**(2V) on a slow device (such as a terminal or pipe or other socket, but not a file) and during a **wait**(2V).

After a **fork**(2V), or **vfork**(2) the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt and reset-signal-handler flags.

The **execve**(2V), call resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt system calls continue to do so.

CODES

The following defines the codes for signals which produce them. All of these symbols are defined in **signal.h**:

Condition	Signal	Code
Sun codes:		
Illegal instruction	**SIGILL**	**ILL_INSTR_FAULT**
Integer division by zero	**SIGFPE**	**FPE_INTDIV_TRAP**
IEEE floating pt inexact	**SIGFPE**	**FPE_FLTINEX_TRAP**
IEEE floating pt division by zero	**SIGFPE**	**FPE_FLTDIV_TRAP**
IEEE floating pt underflow	**SIGFPE**	**FPE_FLTUND_TRAP**
IEEE floating pt operand error	**SIGFPE**	**FPE_FLTOPERR_TRAP**
IEEE floating pt overflow	**SIGFPE**	**FPE_FLTOVF_FAULT**
Hardware bus error	**SIGBUS**	**BUS_HWERR**
Address alignment error	**SIGBUS**	**BUS_ALIGN**
No mapping fault	**SIGSEGV**	**SEGV_NOMAP**
Protection fault	**SIGSEGV**	**SEGV_PROT**
Object error	**SIGSEGV**	**SEGV_CODE**(code)=**SEGV_OBJERR**
Object error number	**SIGSEGV**	**SEGV_ERRNO**(code)
SPARC codes:		
Privileged instruction violation	**SIGILL**	**ILL_PRIVINSTR_FAULT**
Bad stack	**SIGILL**	**ILL_STACK**
Trap #*n* (1 <= *n* <= 127)	**SIGILL**	**ILL_TRAP_FAULT**(*n*)
Integer overflow	**SIGFPE**	**FPE_INTOVF_TRAP**
Tag overflow	**SIGEMT**	**EMT_TAG**
MC680X0 codes:		
Privilege violation	**SIGILL**	**ILL_PRIVVIO_FAULT**
Coprocessor protocol error	**SIGILL**	**ILL_INSTR_FAULT**
Trap #*n* (1 <= *n* <= 14)	**SIGILL**	**ILL_TRAP***n* **_FAULT**
A-line op code	**SIGEMT**	**EMT_EMU1010**
F-line op code	**SIGEMT**	**EMT_EMU1111**
CHK or CHK2 instruction	**SIGFPE**	**FPE_CHKINST_TRAP**
TRAPV or TRAPcc or cpTRAPcc	**SIGFPE**	**FPE_TRAPV_TRAP**

IEEE floating pt compare unordered	**SIGFPE**	**FPE_FLTBSUN_TRAP**
IEEE floating pt signaling NaN	**SIGFPE**	**FPE_FLTNAN_TRAP**

ADDR

The *addr* signal handler parameter is defined as follows:

Signal	Code	Addr
Sun:		
SIGILL	Any	address of faulted instruction
SIGEMT	Any	address of faulted instruction
SIGFPE	Any	address of faulted instruction
SIGBUS	**BUS_HWERR**	address that caused fault
SIGSEGV	Any	address that caused fault
SPARC:		
SIGBUS	**BUS_ALIGN**	address of faulted instruction
MC680X0:		
SIGBUS	**BUS_ALIGN**	address that caused fault

The accuracy of *addr* is machine dependent. For example, certain machines may supply an address that is on the same page as the address that caused the fault. If an appropriate *addr* cannot be computed it will be set to **SIG_NOADDR**.

RETURN VALUES

sigvec( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

sigvec( ) will fail and no new signal handler will be installed if one of the following occurs:

EFAULT Either *vec* or *ovec* is not a NULL pointer and points to memory that is not a valid part of the process address space.

EINVAL *Sig* is not a valid signal number.

An attempt was made to ignore or supply a handler for **SIGKILL** or **SIGSTOP**.

SEE ALSO

execve(2V), **fcntl**(2V), **fork**(2V), **getitimer**(2), **getrlimit**(2), **ioctl**(2), **kill**(2V), **ptrace**(2), **read**(2V), **sigblock**(2), **sigpause**(2V), **sigsetmask**(2), **sigstack**(2), **umask**(2V), **vfork**(2), **wait**(2V), **write**(2V), **setjmp**(3V), **signal**(3V), **streamio**(4), **termio**(4), **win**(4S), **lockd**(8C)

NOTES

SIGPOLL is a synonym for **SIGIO**. A **SIGIO** will be issued when a file descriptor corresponding to a STREAMS (see **intro**(2)) file has a "selectable" event pending. Unless that descriptor has been put into asynchronous mode (see **fcntl (2V)**, a process must specifically request that this signal be sent using the **I_SETSIG ioctl**(2) call (see **streamio**(4)). Otherwise, the process will never receive **SIGPOLL**.

The handler routine can be declared:

```
void handler(sig, code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **signal.h**), used to restore the context from before the signal; and *addr* is additional address information.

Programs that must be portable to UNIX systems other than 4.2BSD should use the **signal**(3V), interface instead.

NAME

socket – create an endpoint for communication

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int socket(domain, type, protocol)
int domain, type, protocol;

DESCRIPTION

socket() creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file **<sys/socket.h>**. The currently understood formats are

PF_UNIX	(UNIX system internal protocols),
PF_INET	(ARPA Internet protocols), and
PF_IMPLINK	(IMP "host at IMP" link layer).

The socket has the indicated *type*, which specifies the semantics of communication. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

A **SOCK_STREAM** type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A **SOCK_DGRAM** socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A **SOCK_SEQPACKET** socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently not implemented for any protocol family. **SOCK_RAW** sockets provide access to internal network interfaces. The types **SOCK_RAW**, which is available only to the super-user, and **SOCK_RDM**, for which no implementation currently exists, are not described here.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see **protocols**(5).

Sockets of type **SOCK_STREAM** are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a **connect**(2) call. Once connected, data may be transferred using **read**(2V) and **write**(2V) calls or some variant of the **send**(2) and **recv**(2) calls. When a session has been completed a **close**(2V), may be performed. Out-of-band data may also be transmitted as described in **send**(2) and received as described in **recv**(2).

The communications protocols used to implement a **SOCK_STREAM** insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with ETIMEDOUT as the specific code in the global variable **errno**. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (for instance 5 minutes). A **SIGPIPE** signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_SEQPACKET sockets employ the same system calls as **SOCK_STREAM** sockets. The only difference is that **read**(2V) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and **SOCK_RAW** sockets allow sending of datagrams to correspondents named in **send**(2) calls. Datagrams are generally received with **recv**(2), which returns the next datagram with its return address.

An **fcntl**(2V) call can be used to specify a process group to receive a **SIGURG** signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events with **SIGIO** signals.

The operation of sockets is controlled by socket level *options*. These options are defined in the file **socket.h**. **getsockopt**(2) and **setsockopt**( ) are used to get and set options, respectively.

RETURN VALUES

socket( ) returns a non-negative descriptor on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

EACCES	Permission to create a socket of the specified type and/or protocol is denied.
EMFILE	The per-process descriptor table is full.
ENFILE	The system file table is full.
ENOBUFS	Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.
EPROTONOSUPPORT	The protocol type or the specified protocol is not supported within this domain.
EPROTOTYPE	The protocol is the wrong type for the socket.

SEE ALSO

accept(2), **bind**(2), **close**(2V), **connect**(2), **fcntl**(2V), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **listen**(2), **read**(2V), **recv**(2), **select**(2), **send**(2), **shutdown**(2), **socketpair**(2), **write**(2V), **protocols**(5)

Network Programming

NAME

socketpair – create a pair of connected sockets

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int socketpair(d, type, protocol, sv)
int d, type, protocol;
int sv[2];

DESCRIPTION

The **socketpair( )** system call creates an unnamed pair of connected sockets in the specified address family *d*, of the specified *type* and using the optionally specified *protocol*. The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are indistinguishable.

RETURN VALUES

socketpair( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EAFNOSUPPORT	The specified address family is not supported on this machine.
EFAULT	The address *sv* does not specify a valid part of the process address space.
EMFILE	Too many descriptors are in use by this process.
EOPNOSUPPORT	The specified protocol does not support creation of socket pairs.
EPROTONOSUPPORT	The specified protocol is not supported on this machine.

SEE ALSO

pipe(2V), **read**(2V), **write**(2V)

BUGS

This call is currently implemented only for the **AF_UNIX** address family.

NAME

stat, lstat, fstat – get file status

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

int stat(path, buf)
char *path;
struct stat *buf;

int lstat(path, buf)
char *path;
struct stat *buf;

int fstat(fd, buf)
int fd;
struct stat *buf;

DESCRIPTION

stat() obtains information about the file named by *path*. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

lstat() is like **stat()** except in the case where the named file is a symbolic link, in which case **lstat()** returns information about the link, while **stat()** returns information about the file the link references.

fstat() obtains the same information about an open file referenced by the argument descriptor, such as would be obtained by an **open**(2V) call.

buf is a pointer to a **stat** structure into which information is placed concerning the file. A **stat** structure includes the following members:

dev_t	**st_dev;**	**/* device file resides on */**
ino_t	**st_ino;**	**/* the file serial number */**
mode_t	**st_mode;**	**/* file mode */**
nlink_t	**st_nlink;**	**/* number of hard links to the file */**
uid_t	**st_uid;**	**/* user ID of owner */**
gid_t	**st_gid;**	**/* group ID of owner */**
dev_t	**st_rdev;**	**/* the device identifier (special files only)*/**
off_t	**st_size;**	**/* total size of file, in bytes */**
time_t	**st_atime;**	**/* file last access time */**
time_t	**st_mtime;**	**/* file last modify time */**
time_t	**st_ctime;**	**/* file last status change time */**
long	**st_blksize;**	**/* preferred blocksize for file system I/O*/**
long	**st_blocks;**	**/* actual number of blocks allocated */**

st_atime Time when file data was last accessed. This can also be set explicitly by **utimes**(2). **st_atime** is not updated for directories searched during pathname resolution.

st_mtime Time when file data was last modified. This can also be set explicitly by **utimes**(2). It is not set by changes of owner, group, link count, or mode.

st_ctime Time when file status was last changed. It is set both both by writing and changing the file status information, such as changes of owner, group, link count, or mode.

The following macros test whether a file is of the specified type. The value *m* is the value of **st_mode**. Each macro evaluates to a non-zero value if the test is true or to zero if the test is false.

S_ISDIR(*m*) Test for directory file.

S_ISCHR(*m*) Test for character special file.

S_ISBLK(*m*) Test for block special file.

S_ISREG(*m*) Test for regular file.

S_ISLNK(*m*) Test for a symbolic link.

S_ISSOCK(*m*) Test for a socket.

S_ISFIFO(*m*) Test for pipe or FIFO special file.

The status information word **st_mode** is bit-encoded using the following masks and bits:

S_IRWXU Read, write, search (if a directory), or execute (otherwise) permissions mask for the owner of the file.

- **S_IRUSR** Read permission bit for the owner of the file.
- **S_IWUSR** Write permission bit for the owner of the file.
- **S_IXUSR** Search (if a directory) or execute (otherwise) permission bit for the owner of the file.

S_IRWXG Read, write, search (if directory), or execute (otherwise) permissions mask for the file group class.

- **S_IRGRP** Read permission bit for the file group class.
- **S_IWGRP** Write permission bit for the file group class.
- **S_IXGRP** Search (if a directory) or execute (otherwise) permission bit for the file group class.

S_IRWXO Read, write, search (if a directory), or execute (otherwise) permissions mask for the file other class.

- **S_IROTH** Read permission bit for the file other class.
- **S_IWOTH** Write permission bit for the file other class.
- **S_IXOTH** Search (if a directory) or execute (otherwise) permission bit for the file other class.

S_ISUID Set user ID on execution. The process's effective user ID is set to that of the owner of the file when the file is run as a program (see **execve**(2V)). On a regular file, this bit should be cleared on any write.

S_ISGID Set group ID on execution. The process's effective group ID is set to that of the file when the file is run as a program (see **execve**(2V)). On a regular file, this bit should be cleared on any write.

In addition, the following bits and masks are made available for backward compatibility:

```
#define  S_IFMT      0170000   /* type of file */
#define  S_IFIFO     0010000   /* FIFO special */
#define  S_IFCHR     0020000   /* character special */
#define  S_IFDIR     0040000   /* directory */
#define  S_IFBLK     0060000   /* block special */
#define  S_IFREG     0100000   /* regular file */
#define  S_IFLNK     0120000   /* symbolic link */
#define  S_IFSOCK    0140000   /* socket */
#define  S_ISVTX     0001000   /* save swapped text even after use */
#define  S_IREAD     0000400   /* read permission, owner */
#define  S_IWRITE    0000200   /* write permission, owner */
#define  S_IEXEC     0000100   /* execute/search permission, owner */
```

For more information on **st_mode** bits see **chmod**(2V).

RETURN VALUES

stat(), **lstat()** and **fstat()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

stat() and **lstat()** will fail if one or more of the following are true:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*buf* or *path* points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.n
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.

fstat() will fail if one or more of the following are true:

EBADF	*fd* is not a valid open file descriptor.
EFAULT	*buf* points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to an empty string.

WARNINGS

The **st_atime** and **st_mtime** fields of the **stat()** are *not* contiguous. Programs that depend on them being contiguous (in calls to **utimes**(2) or **utime**(3V)) will not work.

SEE ALSO

chmod(2V), **chown**(2V), **link**(2V), **open**(2V), **read**(2V), **readlink**(2), **rename**(2V), **truncate**(2), **unlink**(2V), **utimes**(2), **write**(2V)

NAME

statfs, fstatfs – get file system statistics

SYNOPSIS

```
#include <sys/vfs.h>

int statfs(path, buf)
char *path;
struct statfs *buf;

int fstatfs(fd, buf)
int fd;
struct statfs *buf;
```

DESCRIPTION

statfs() returns information about a mounted file system. *path* is the path name of any file within the mounted filesystem. *buf* is a pointer to a **statfs()** structure defined as follows:

```
typedef struct {
        long    val[2];
} fsid_t;

struct statfs {
        long    f_type;     /* type of info, zero for now */
        long    f_bsize;    /* fundamental file system block size */
        long    f_blocks;   /* total blocks in file system */
        long    f_bfree;    /* free blocks */
        long    f_bavail;   /* free blocks available to non-super-user */
        long    f_files;    /* total file nodes in file system */
        long    f_ffree;    /* free file nodes in fs */
        fsid_t  f_fsid;     /* file system id */
        long    f_spare[7]; /* spare for later */
};
```

Fields that are undefined for a particular file system are set to −1. **fstatfs()** returns the same information about an open file referenced by descriptor *fd*.

RETURN VALUES

statfs() and **fstatfs()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

statfs() fails if one or more of the following are true:

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*buf* or *path* points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.

fstatfs( ) fails if one or more of the following are true:

EBADF	*fd* is not a valid open file descriptor.
EFAULT	*buf* points to an invalid address.
EIO	An I/O error occurred while reading from the file system.

BUGS

The NFS revision 2 protocol does not permit the number of free files to be provided to the client; thus, when **statfs( )** or **fstatfs( )** are done on a file on an NFS file system, **f_files** and **f_ffree** are always −1.

NAME

swapon – add a swap device for interleaved paging/swapping

SYNOPSIS

int swapon(special)
char *special;

DESCRIPTION

swapon( ) makes the block device *special* available to the system for allocation for paging and swapping. The names of potentially available devices are known to the system and defined at system configuration time. The size of the swap area on *special* is calculated at the time the device is first made available for swapping.

RETURN VALUES

swapon( ) returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *special*.
EBUSY	The device referred to by *special* has already been made available for swapping.
EFAULT	*special* points outside the process's address space.
EIO	An I/O error occurred while reading from or writing to the file system.
	An I/O error occurred while opening the swap device.
ELOOP	Too many symbolic links were encountered in translating *special*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENODEV	The device referred to by *special* was not configured into the system as a swap device.
ENOENT	The device referred to by *special* does not exist.
ENOTBLK	The file referred to by *special* is not a block device.
ENOTDIR	A component of the path prefix of *special* is not a directory.
ENXIO	The major device number of the device referred to by *special* is out of range (this indicates no device driver exists for the associated hardware).
EPERM	The caller is not the super-user.

SEE ALSO

fstab(5), **config**(8), **swapon**(8)

BUGS

There is no way to stop swapping on a disk so that the pack may be dismounted.

This call will be upgraded in future versions of the system.

NAME

symlink – make symbolic link to a file

SYNOPSIS

int symlink(name1, name2)
char *name1, *name2;

DESCRIPTION

A symbolic link *name2* is created to *name1* (*name2* is the name of the file created, *name1* is the string used in creating the symbolic link). Either name may be an arbitrary path name; the files need not be on the same file system.

The file that the symbolic link points to is used when an **open**(2V) operation is performed on the link. A **stat**(2V), on a symbolic link returns the linked-to file, while an **lstat**( ) (refer to **stat**(2V)) returns information about the link itself. This can lead to surprising results when a symbolic link is made to a directory. To avoid confusion in programs, the **readlink**(2) call can be used to read the contents of a symbolic link.

RETURN VALUES

symlink( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

The symbolic link is made unless one or more of the following are true:

EACCES	Search permission is denied for a component of the path prefix of *name2*.
EDQUOT	The directory in which the entry for the new symbolic link is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
	The new symbolic link cannot be created because the user's quota of disk blocks on the file system which will contain the link has been exhausted.
	The user's quota of inodes on the file system on which the file is being created has been exhausted.
EEXIST	The file referred to by *name2* already exists.
EFAULT	*name1* or *name2* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *name2*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	A component of the path prefix of *name2* does not exist.
ENOSPC	The directory in which the entry for the new symbolic link is being placed cannot be extended because there is no space left on the file system containing the directory.
	The new symbolic link cannot be created because there is no space left on the file system which will contain the link.
	There are no free inodes on the file system on which the file is being created.
ENOTDIR	A component of the path prefix of *name2* is not a directory.
EROFS	The file *name2* would reside on a read-only file system.

SEE ALSO

ln(1V), **link**(2V), **readlink**(2), **unlink**(2V)

NAME

sync – update super-block

SYNOPSIS

sync()

DESCRIPTION

sync() writes out all information in core memory that should be on disk. This includes modified super blocks, modified inodes, and delayed block I/O.

sync() should be used by programs that examine a file system, for example **fsck**(8), **df**(1V), etc. **sync()** is mandatory before a boot.

SEE ALSO

fsync(2), **cron**(8)

BUGS

The writing, although scheduled, is not necessarily complete upon return from **sync()**.

NAME

syscall – indirect system call

SYNOPSIS

#include <sys/syscall.h>

int syscall(number[, arg, ...])
int number;

DESCRIPTION

syscall() performs the system call whose assembly language interface has the specified *number*, and arguments *arg* .... Symbolic constants for system calls can be found in the header file **<sys/syscall.h>.**

RETURN VALUES

syscall() returns the return value of the system call specified by *number*.

SEE ALSO

intro(2), **pipe**(2V)

WARNINGS

There is no way to use **syscall()** to call functions such as **pipe**(2V), which return values that do not fit into one hardware register.

Since many system calls are implemented as library wrappers around traps to the kernel, these calls may not behave as documented when called from **syscall()**, which bypasses these wrappers. For these reasons, using **syscall()** is not recommended.

NAME

sysconf – query system related limits, values, options

SYNOPSIS

#include <unistd.h>

long sysconf(name)
int name;

DESCRIPTION

The **sysconf**( ) function provides a method for the application to determine the current value of a configurable system limit or option (variable). The value does not change during the lifetime of the calling process.

The convention used throughout sections 2 and 3 is that {LIMIT} means that LIMIT is something that can change from system to system and applications that want accurate values need to call **sysconf**( ). These values are things that have been historically available in header files such as **<sys/param.h>**.

The following lists the conceptual name and meaning of each variable.

Name	*Meaning*
{ARG_MAX}	Max combined size of **argv**[ ] & **envp**[ ].
{CHILD_MAX}	Max processes allowed to any UID.
{CLK_TCK}	Ticks per second (**clock_t**).
{NGROUPS_MAX}	Max simultaneous groups one may belong to.
{OPEN_MAX}	Max open files per process.
{_POSIX_JOB_CONTROL}	Job control supported (boolean).
{_POSIX_SAVED_IDS}	Saved ids (**seteuid**( )) supported (boolean).
{_POSIX_VERSION}	Version of the POSIX.1 standard supported.

The following table lists the conceptual name of each variable and the flag passed to **sysconf**( ) to retrieve the value of each variable.

Name	*Sysconf flag*
{ARG_MAX}	**_SC_ARG_MAX**
{CHILD_MAX}	**_SC_CHILD_MAX**
{CLK_TCK}	**_SC_CLK_TCK**
{NGROUPS_MAX}	**_SC_NGROUPS_MAX**
{OPEN_MAX}	**_SC_OPEN_MAX**
{_POSIX_JOB_CONTROL}	**_SC_JOB_CONTROL**
{_POSIX_SAVED_IDS}	**_SC_SAVED_IDS**
{_POSIX_VERSION}	**_SC_VERSION**

RETURN VALUES

sysconf( ) returns the current variable value on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

EINVAL The value of *name* is invalid.

NAME

truncate, ftruncate – set a file to a specified length

SYNOPSIS

```
#include <sys/types.h>

int truncate(path, length)
char *path;
off_t length;

int ftruncate(fd, length)
int fd;
off_t length;
```

DESCRIPTION

truncate( ) causes the file referred to by *path* (or for **ftruncate( )** the object referred to by *fd*) to have a size equal to *length* bytes. If the file was previously longer than *length*, the extra bytes are removed from the file. If it was shorter, bytes between the old and new lengths are read as zeroes. With **ftruncate( )**, the file must be open for writing.

RETURN VALUES

truncate( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

truncate( ) may set **errno** to:

EACCES	Search permission is denied for a component of the path prefix of *path*.
	Write permission is denied for the file referred to by *path*.
EFAULT	*path* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
EISDIR	The file referred to by *path* is a directory.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EROFS	The file referred to by *path* resides on a read-only file system.

ftruncate( ) may set **errno** to:

EINVAL	*fd* is not a valid descriptor of a file open for writing.
	fd refers to a socket, not to a file.
EIO	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

open(2V)

BUGS

These calls should be generalized to allow ranges of bytes in a file to be discarded.

NAME

umask – set file creation mode mask

SYNOPSIS

#include <sys/stat.h>

int umask(mask)
int mask;

SYSTEM V SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(mask)
mode_t mask;

DESCRIPTION

umask() sets the process's file creation mask to *mask* and returns the previous value of the mask. The low-order 9 bits of *mask* are used whenever a file is created, clearing corresponding bits in the file access permissions. (see **stat**(2V)). This clearing restricts the default access to a file.

The mask is inherited by child processes.

RETURN VALUES

umask() returns the previous value of the file creation mask.

SEE ALSO

chmod(2V), **mknod**(2V), **open**(2V)

NAME

uname – get information about current system

SYNOPSIS

#include <sys/utsname.h>

int uname (name)
struct utsname *name;

DESCRIPTION

uname( ) stores information identifying the current operating system in the structure pointed to by *name*.

uname( ) uses the structure defined in **<sys/utsname.h>**, the members of which are:

```
struct utsname {
        char    sysname[9];
        char    nodename[9];
        char    nodeext[65-9];
        char    release[9];
        char    version[9];
        char    machine[9];
}
```

uname( ) places a null-terminated character string naming the current operating system in the character array *sysname*; this string is "SunOS" on Sun systems. *nodename* is set to the name that the system is known by on a communications network; this is the same value as is returned by **gethostname**(2). *release* and *version* are set to values that further identify the operating system. *machine* is set to a standard name that identifies the hardware on which the SunOS system is running. This is the same as the value displayed by **arch**(1).

RETURN VALUES

uname( ) returns:

0 on success.

–1 on failure.

SEE ALSO

arch(1), **uname**(1), **gethostname**(2)

NOTES

nodeext is provided for backwards compatability with previous SunOS Releases and provides space for node names longer than eight bytes. Applications should not use *nodeext*. To be maximally portable, applications that want to copy the node name to another string should use **strlen(nodename)** rather than the constant 9 or **sizeof(nodename)** as the size of the target string.

System administrators should note that systems with node names longer than eight bytes do not conform to *IEEE Std 1003.1-1988*, *System V Interface Definition* (Issue 2), or *X/Open Portability Guide* (Issue 2) requirements.

NAME

unlink – remove directory entry

SYNOPSIS

int unlink(path)
char *path;

DESCRIPTION

unlink() removes the directory entry named by the pathname pointed to by *path* and decrements the link count of the file referred to by that entry. If this entry was the last link to the file, and no process has the file open, then all resources associated with the file are reclaimed. If, however, the file was open in any process, the actual resource reclamation is delayed until it is closed, even though the directory entry has disappeared.

If *path* refers to a directory, the effective user-ID of the calling process must be super-user.

Upon successful completion, **unlink()** marks for update the **st_ctime** and **st_mtime** fields of the parent directory. Also, if the file's link count is not zero, the **st_ctime** field of the file is marked for update.

RETURN VALUES

unlink() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
	Write permission is denied for the directory containing the link to be removed.
EBUSY	The entry to be unlinked is the mount point for a mounted file system.
EFAULT	*path* points outside the process's allocated address space.
EINVAL	The file referred to by *path* is the current directory, '.'.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EPERM	The file referred to by *path* is a directory and the effective user ID of the process is not the super-user.
EROFS	The file referred to by *path* resides on a read-only file system.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to an empty string.

SEE ALSO

close(2V), **link**(2V), **rmdir**(2V)

NOTES

Applications should use **rmdir**(2V) to remove directories. Although **root** may use **unlink()** on directories, all users may use **rmdir()**.

NAME

unmount, umount – remove a file system

SYNOPSIS

int unmount(name)
char *name;

SYSTEM V SYNOPSIS

int umount(special)
char *special;

DESCRIPTION

unmount() announces to the system that the directory *name* is no longer to refer to the root of a mounted file system. The directory *name* reverts to its ordinary interpretation.

Only the super-user may call **unmount()**.

SYSTEM V DESCRIPTION

umount() reqests that a previously mounted file system contained on the block special device referred to by *special* be unmounted. *special* points to a path name. After the file system is unmounted, the directory on which it was mounted reverts to its ordinary interpretation.

Only the super-user may call **umount()**.

Note: Unlike the path name argument to **unmount()** which refers to the directory on which the file system is mounted, *special* refers to the block special device containing the mounted file system itself.

RETURN VALUES

unmount() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

SYSTEM V RETURN VALUES

umount() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix.
EBUSY	A process is holding a reference to a file located on the file system.
EFAULT	*name* points outside the process's allocated address space.
EINVAL	*name* is not the root of a mounted file system.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating the path name.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	*name* does not exist.
ENOTDIR	A component of the path prefix of *name* is not a directory.
EPERM	The caller is not the super-user.

SYSTEM V ERRORS

EINVAL	The device referred to by *special* is not mounted.
ENOENT	The named file does not exist.

ENOTBLK	*special* does not refer to a block special file.
ENOTDIR	A component of the path prefix of *special* is not a directory.
ENXIO	The device referred to by *special* does not exist.

SEE ALSO

mount(2V), **mount**(8)

BUGS

The error codes are in a state of disarray; too many errors appear to the caller as one value.

NAME

ustat – get file system statistics

SYNOPSIS

```
#include <sys/types.h>
#include <ustat.h>

int ustat(dev, buf)
dev_t dev;
struct ustat *buf;
```

DESCRIPTION

ustat() returns information about a mounted file system. *dev* is a device number identifying a device containing a mounted file system. This is normally the value returned in the **st_dev** field of a **stat** structure when a **stat()**, **fstat()**, or **lstat()** call is made on a file on that file system. *buf* is a pointer to a **ustat** structure that includes the following elements:

```
daddr_t  f_tfree;      /* Total blocks available to non-super-user */
ino_t    f_tinode;     /* Number of free files */
char     f_fname[6];   /* Filsys name */
char     f_fpack[6];   /* Filsys pack name */
```

The **f_fname** and **f_fpack** fields are always set to a null string. Other fields that are undefined for a particular file system are set to −1.

RETURN VALUES

ustat() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EFAULT *buf* points to an invalid address.

EINVAL *dev* is not the device number of a device containing a mounted file system.

EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

stat(2V), **statfs**(2)

BUGS

The NFS revision 2 protocol does not permit the number of free files to be provided to the client; thus, when **ustat()** is done on an NFS file system, **f_tinode** is always −1.

NAME

utimes – set file times

SYNOPSIS

#include <sys/types.h>

int utimes(file, tvp)
char *file;
struct timeval *tvp;

DESCRIPTION

utimes( ) sets the access and modification times of the file named by *file*.

If *tvp* is NULL, the access and modification times are set to the current time. A process must be the owner of the file or have write permission for the file to use **utimes( )** in this manner.

If *tvp* is not NULL, it is assumed to point to an array of two **timeval** structures. The access time is set to the value of the first member, and the modification time is set to the value of the second member. Only the owner of the file or the super-user may use **utimes( )** in this manner.

In either case, the *inode-changed* time of the file is set to the current time.

RETURN VALUES

utimes( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *file*.
EACCES	The effective user ID of the process is not super-user and not the owner of the file, write permission is denied for the file, and *tvp* is NULL.
EFAULT	*file* or *tvp* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *file*.
ENOENT	The file referred to by *file* does not exist.
ENOTDIR	A component of the path prefix of *file* is not a directory.
EPERM	The effective user ID of the process is not super-user and not the owner of the file, and *tvp* is not NULL.
EROFS	The file system containing the file is mounted read-only.

SEE ALSO

stat(2V)

NAME

vadvise – give advice to paging system

SYNOPSIS

#include <sys/vadvise.h>

vadvise(param)
int param;

DESCRIPTION

vadvise( ) is used to inform the system that process paging behavior merits special consideration. Parameters to **vadvise**( ) are defined in the file **<sys/vadvise.h>**. Currently, two calls to **vadvise**( ) are implemented.

vadvise(VA_ANOM);

advises that the paging behavior is not likely to be well handled by the system's default algorithm, since reference information that is collected over macroscopic intervals (for instance, 10-20 seconds) will not serve to indicate future page references. The system in this case will choose to replace pages with little emphasis placed on recent usage, and more emphasis on referenceless circular behavior. It is *essential* that processes which have very random paging behavior (such as LISP during garbage collection of very large address spaces) call **vadvise**, as otherwise the system has great difficulty dealing with their page-consumptive demands.

vadvise(VA_NORM);

restores default paging replacement behavior after a call to

vadvise(VA_ANOM);

BUGS

The current implementation of **vadvise**( ) will go away soon, being replaced by a per-page **vadvise**( ) facility.

NAME

vfork – spawn new process in a virtual memory efficient way

SYNOPSIS

#include <vfork.h>

int vfork( )

DESCRIPTION

vfork( ) can be used to create new processes without fully copying the address space of the old process, which is horrendously inefficient in a paged environment. It is useful when the purpose of **fork**(2V), would have been to create a new system context for an **execve**(2V). **vfork( )** differs from **fork( )** in that the child borrows the parent's memory and thread of control until a call to **execve**(2V), or an exit (either by a call to **exit**(2V) or abnormally.) The parent process is suspended while the child is using its resources.

vfork( ) returns 0 in the child's context and (later) the process ID (PID) of the child in the parent's context.

vfork( ) can normally be used just like **fork**. It does not work, however, to return while running in the child's context from the procedure which called **vfork( )** since the eventual return from **vfork( )** would then return to a no longer existent stack frame. Be careful, also, to call **_exit( )** rather than **exit( )** if you cannot *execve*, since **exit( )** will flush and close standard I/O channels, and thereby mess up the parent processes standard I/O data structures. (Even with **fork( )** it is wrong to call **exit( )** since buffered data would then be flushed twice.)

On Sun-4 machines, the parent inherits the values of local and incoming argument registers from the child. Since this violates the usual data flow properties of procedure calls, the file **<vfork.h>** must be included in programs that are compiled using global optimization.

RETURN VALUES

On success, **vfork( )** returns 0 to the child process and returns the process ID of the child process to the parent process. On failure, **vfork( )** returns –1 to the parent process, sets **errno** to indicate the error, and no child process is created.

SEE ALSO

execve(2V), **exit**(2V), **fork**(2V), **ioctl**(2), **sigvec**(2), **wait**(2V)

BUGS

This system call will be eliminated in a future release. System implementation changes are making the efficiency gain of **vfork( )** over **fork**(2V) smaller. The memory sharing semantics of **vfork( )** can be obtained through other mechanisms.

To avoid a possible deadlock situation, processes that are children in the middle of a **vfork( )** are never sent **SIGTTOU** or **SIGTTIN** signals; rather, output or *ioctl*s are allowed and input attempts result in an EOF indication.

NAME

vhangup – virtually ''hangup'' the current control terminal

SYNOPSIS

vhangup()

DESCRIPTION

vhangup() is used by the initialization process **init**(8) (among others) to arrange that users are given "clean" terminals at login, by revoking access of the previous users' processes to the terminal. To affect this, **vhangup()** searches the system tables for references to the control terminal of the invoking process, revoking access permissions on each instance of the terminal that it finds. Further attempts to access the terminal by the affected processes will yield I/O errors (EBADF). Finally, a **SIGHUP** (hangup signal) is sent to the process group of the control terminal.

SEE ALSO

init(8)

BUGS

Access to the control terminal using **/dev/tty** is still possible.

This call should be replaced by an automatic mechanism that takes place on process exit.

NAME

wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED, WEXITSTATUS, WTERMSIG, WSTOPSIG – wait for process to terminate or stop, examine returned status

SYNOPSIS

```
#include <sys/wait.h>

int wait(statusp)
int *statusp;

int waitpid(pid, statusp, options)
int pid;
int *statusp;
int options;

#include <sys/time.h>
#include <sys/resource.h>

int wait3(statusp, options, rusage)
int *statusp;
int options;
struct rusage *rusage;

int wait4(pid, statusp, options, rusage)
int pid;
int *statusp;
int options;
struct rusage *rusage;

WIFSTOPPED(status)
int status;

WIFSIGNALED(status)
int status;

WIFEXITED(status)
int status

WEXITSTATUS(status)
int status

WTERMSIG(status)
int status

WSTOPSIG(status)
int status
```

SYSTEM V SYNOPSIS

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(statusp)
int *statusp;

pid_t waitpid(pid, statusp, options)
pid_t pid;
int *statusp;
int options;
```

DESCRIPTION

wait( ) delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child has died or stopped due to tracing and this has not been reported using **wait( )**, return is immediate, returning the process ID and exit status of one of those children. If that child had died, it is discarded. If there are no children, return is immediate with the value −1 returned. If there are only running or stopped but reported children, the calling process is blocked.

If *statusp* is not a NULL pointer, then on return from a successful **wait( )** call the status of the child process whose process ID is the return value of **wait( )** is stored in the location pointed to by *statusp*. It indicates the cause of termination and other information about the terminated process in the following manner:

- If the first byte (the low-order 8 bits) are equal to 0177, the child process has stopped. The next byte contains the number of the signal that caused the process to stop. See **ptrace**(2) and **sigvec**(2).
- If the first byte (the low-order 8 bits) are non-zero and are not equal to 0177, the child process terminated due to a signal. The low-order 7 bits contain the number of the signal that terminated the process. In addition, if the low-order seventh bit (that is,·bit 0200) is set, a ''core image'' of the process was produced (see **sigvec**(2)).
- Otherwise, the child process terminated due to a call to **exit**(2V). The next byte contains the low-order 8 bits of the argument that the child process passed to **exit( )**.

waitpid( ) behaves identically to **wait( )** if *pid* has a value of −1 and *options* has a value of zero. Otherwise, the behavior of **waitpid( )** is modified by the values of *pid* and *options* as follows:

pid specifies a set of child processes for which status is requested. **waitpid( )** only returns the status of a child process from this set.

- If *pid* is equal to −1, status is requested for any child process. In this repect, **waitpid( )** is then equivalent to **wait( )**.
- If *pid* is greater than zero, it specifies the process ID of a single child process for which status is requested.
- If *pid* is equal to zero, status is requested for any child process whose process group ID is equal to that of the calling process.
- If *pid* is less than −1, status is requested for any child process whose process group ID is equal to the absolute value of *pid*.

options is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header **<sys/wait.h>**:

WNOHANG
: **waitpid( )** does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by *pid*.

WUNTRACED
: The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.

wait3( ) is an alternate interface that allows both non-blocking status collection and the collection of the status of children stopped by any means. The *status* parameter is defined as above. The *options* parameter is used to indicate the call should not block if there are no processes that have status to report (**WNOHANG**), and/or that children of the current process that are stopped due to a **SIGTTIN**, **SIGTTOU**, **SIGTSTP**, or **SIGSTOP** signal are eligible to have their status reported as well (**WUNTRACED**). A terminated child is discarded after it reports status, and a stopped process will not report its status more than once. If *rusage* is not a NULL pointer, a summary of the resources used by the terminated process and all its children is returned. (This information is currently not available for stopped processes.)

When the **WNOHANG** option is specified and no processes have status to report, **wait3()** returns 0. The **WNOHANG** and **WUNTRACED** options may be combined by ORing the two values.

wait4() is another alternate interface. With a *pid* argument of 0, it is equivalent to **wait3()**. If *pid* has a nonzero value, then **wait4()** returns status only for the indicated process ID, but not for any other child processes.

WIFSTOPPED, **WIFSIGNALED**, **WIFEXITED**, **WEXITSTATUS**, **WTERMSIG**, and **WSTOPSIG** are macros that take an argument *status*, of type **'int'**, as returned by **wait()**, **wait3()**, or **wait4()**. **WIFSTOPPED** evaluates to true (1) when the process for which the **wait()** call was made is stopped, or to false (0) otherwise. If **WIFSTOPPED**(*status*) is non-zero, **WSTOPSIG** evaluates to the number of the signal that caused the child process to stop. **WIFSIGNALED** evaluates to true when the process was terminated with a signal. If **WIFSIGNALED**(*status*) is non-zero, **WTERMSIG** evaluates to the number of the signal that caused the termination of the child process. **WIFEXITED** evaluates to true when the process exited by using an **exit**(2V) call. If **WIFEXITED**(*status*) is non-zero, **WEXITSTATUS** evaluates to the low-order byte of the argument that the child process passed to **_exit()** (see **exit**(2V)) or **exit**(3), or the value the child process returned from **main()** (see **execve**(2V)).

If the information stored at the location pointed to by *statusp* was stored there by a call to **waitpid()** that specified the **WUNTRACED** flag, exactly one of the macros **WIFEXITED**(**statusp*), **WIFSIGNALED**(**statusp*), and **WIFSTOPPED**(**statusp*) will evaluate to a non-zero value. If the information stored at the location pointed to by *statusp* was stored there by a call to **waitpid()** that did *not* specify the **WUNTRACED** flag or by a call to **wait()**, exactly one of the macros **WIFEXITED**(**statusp*) and **WIFSIGNALED**(**statusp*) will evaluate to a non-zero value.

If a parent process terminates witout waiting for all of its child processes to terminate, the remaining child processes are assigned the parent process ID of 1, corresponding to **init**(8).

RETURN VALUES

If **wait()** or **waitpid()** returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

If **wait()** or **waitpid()** return due to the delivery of a signal to the calling process, a value of −1 is returned and **errno** is set to EINTR. If **waitpid()** function was invoked with **WNOHANG** set in *options*, it has at least one child process specified by *pid* for which status is not available, and status is not available for any process specified by *pid*, a value of zero is returned. Otherwise, a value of −1 is returned, and **errno** is set to indicate the error.

wait3() and **wait4()** return 0 if **WNOHANG** is specified and there are no stopped or exited children, and return the process ID of the child process if they return due to a stopped or terminated child process. Otherwise, they return a value of −1 and set **errno** to indicate the error.

ERRORS

wait(), **wait3()**, or **wait4()** will fail and return immediately if one or more of the following are true:

ECHILD The calling process has no existing unwaited-for child processes.

EFAULT *statusp* or *rusage* points to an illegal address.

EINTR The function was interrupted by a signal. The value of the location pointed to by *statusp* is undefined.

waitpid() may set **errno** to:

ECHILD The process or process group specified by *pid* does not exist or is not a child of the calling process.

EINTR The function was interrupted by a signal. The value of the location pointed to by *statusp* is undefined.

EINVAL The value of *options* is not valid.

wait( ), **wait3( )**, and **wait4( )** will terminate prematurely, return −1, and set **errno** to: EINTR upon the arrival of a signal whose **SV_INTERRUPT** bit in its flags field is set (see **sigvec**(2) and **siginterrupt**(3V)). **signal**(3V), in the System V compatibility library, sets this bit for any signal it catches.

SEE ALSO

exit(2V), **fork**(2V), **getrusage**(2), **ptrace**(2), **sigvec**(2), **pause**(3V), **siginterrupt**(3V), **signal**(3V), **times**(3V)

NOTES

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

wait( ), **wait3( )**, and **wait4( )** are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the **SV_INTERRUPT** bit is set in the flags for that signal.

Previous SunOS releases used **union wait *statusp** and **union wait status** in place of **int *statusp** and **int**status. The union contained a member **w_status** that could be treated in the same way as *status*.

Other members of the **wait** union could be used to extract this information more conveniently:

- If the **w_stopval** member had the value **WSTOPPED**, the child process had stopped; the value of the **w_stopsig** member was the signal that stopped the process.
- If the **w_termsig** member was non-zero, the child process terminated due to a signal; the value of the **w_termsig** member was the number of the signal that terminated the process. If the **w_coredump** member was non-zero, a core dump was produced.
- Otherwise, the child process terminated due to a call to **exit( )**. The value of the **w_retcode** member was the low-order 8 bits of the argument that the child process passed to **exit( )**.

union wait is obsolete in light of the new specifications provided by *IEEE Std 1003.1-1988* and endorsed by *SVID89* and *XPG3*. SunOS Release 4.1 supports **union wait** for backward compatibility, but it will disappear in a future release.

NAME

write, writev – write output

SYNOPSIS

```
int write(fd, buf, nbyte)
int fd;
char *buf;
int nbyte;

#include <sys/types.h>
#include <sys/uio.h>

int writev(fd, iov, iovcnt)
int fd;
struct iovec *iov;
int iovcnt;
```

SYSTEM V SYNOPSIS

```
int write(fd, buf, nbyte)
int fd;
char *buf;
unsigned nbyte;
```

DESCRIPTION

write() attempts to write *nbyte* bytes of data to the object referenced by the descriptor *fd* from the buffer pointed to by *buf*. **writev()** performs the same action, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0],*iov*[1], ..., *iov*[*iovcnt* – 1]. If *nbyte* is zero, **write()** takes no action and returns 0. **writev()**, however, returns –1 and sets the global variable **errno** (see ERRORS below).

For **writev()**, the **iovec** structure is defined as

```
struct iovec {
        caddr_t iov_base;
        int     iov_len;
};
```

Each **iovec** entry specifies the base address and length of an area in memory from which data should be written. **writev()** always writes a complete area before proceeding to the next.

On objects capable of seeking, the **write()** starts at a position given by the seek pointer associated with *fd*, (see **lseek**(2V)). Upon return from **write()**, the seek pointer is incremented by the number of bytes actually written.

Objects that are not capable of seeking always write from the current position. The value of the seek pointer associated with such an object is undefined.

If the **O_APPEND** flag of the file status flags is set, the seek pointer is set to the end of the file prior to each write.

If the process calling **write()** or **writev()** receives a signal before any data are written, the system call is restarted, unless the process explicitly set the signal to interrupt the call using **sigvec()** or **sigaction()** (see the discussions of **SV_INTERRUPT** on **sigvec**(2) and **SA_INTERRUPT** on **sigaction**(3V)). If **write()** or **writev()** is interrupted by a signal after successfully writing some data, it returns the number of bytes written.

For regular files, if the **O_SYNC** flag of the file status flags is set, **write()** does not return until both the file data and file status have been physically updated. This function is for special applications that require extra reliability at the cost of performance. For block special files, if **O_SYNC** is set, the **write()** does not return until the data has been physically updated.

If the real user is not the super-user, then **write()** clears the set-user-id bit on a file. This prevents penetration of system security by a user who “captures” a writable set-user-id file owned by the super-user.

For STREAMS (see **intro**(2)) files, the operation of **write()** and **writev()** are determined by the values of the minimum and maximum packet sizes accepted by the *stream*. These values are contained in the topmost *stream* module. Unless the user pushes (see **I_PUSH** in **streamio**(4)) the topmost module, these values can not be set or tested from user level. If the total number of bytes to be written falls within the packet size range, that many bytes are written. If the total number of bytes to be written does not fall within the range and the minimum packet size value is zero, **write()** and **writev()** break the data to be written into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If the total number of bytes to be written does not fall within the range and the minimum value is non-zero, **write()** and **writev()** fail and set **errno** to ERANGE. Writing a zero-length buffer (the total number of bytes to be written is zero) sends zero bytes with zero returned.

When a descriptor or the object it refers to is marked for non-blocking I/O, and the descriptor refers to an object subject to flow control, such as a socket, a pipe (or FIFO), or a *stream*, **write()** and **writev()** may write fewer bytes than requested; the return value must be noted, and the remainder of the operation should be retried when possible. If such an object's buffers are full, so that it cannot accept any data, then:

- If the object to which the descriptor refers is marked for non-blocking I/O using the **FIONBIO** request to **ioctl**(2), or by using **fcntl**(2V) to set the **FNDELAY** or **O_NDELAY** flag (defined in **<sys/fcntl.h>**), **write()** returns −1 and sets **errno** to EWOULDBLOCK.

Upon successful completion, **write()** marks for update the **st_ctime** and **st_mtime** fields of the file.

SYSTEM V DESCRIPTION

write() and **writev()** behave as described above, except:

When a descriptor or the object it refers to is marked for non-blocking I/O, and the descriptor refers to an object subject to flow control, such as a socket, a pipe (or FIFO), or a *stream*, **write()** and **writev()** may write fewer bytes than requested; the return value must be noted, and the remainder of the operation should be retried when possible. If such an object's buffers are full, so that it cannot accept any data, then:

- If the descriptor is marked for non-blocking I/O by using **fcntl()** to set the **FNBIO** or **O_NDELAY** flag (defined in **<sys/fcntl.h>**), and does not refer to a *stream*, the **write()** returns 0. If the descriptor is marked for non-blocking I/O, and refers to a *stream*, **write()** returns −1 and sets **errno** to EAGAIN.
- If the descriptor is marked for non-blocking I/O using **fcntl()** to set the **FNONBLOCK** or **O_NONBLOCK** flag (defined in **<sys/fcntl.h>**), **write()** requests for {PIPE_BUF} (see **pathconf**(2V)) or fewer bytes either succeed completely and return *nbyte*, or return −1 and set **errno** to EAGAIN. A **write()** request for greater than {PIPE_BUF} bytes either transfers what it can and returns the number of bytes written, or transfers no data and returns −1 and sets **errno** to EAGAIN. If a **write()** request is greater than {PIPE_BUF} bytes and all data previously written to the pipe has been read, **write()** transfers at least {PIPE_BUF} bytes.

RETURN VALUES

write() and **writev()** return the number of bytes actually written on success. On failure, they return −1 and set **errno** to indicate the error.

ERRORS

write() and **writev()** fail and the seek pointer remains unchanged if one or more of the following are true:

EBADF	*fd* is not a valid descriptor open for writing.
EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.
EFAULT	Part of *iov* or data to be written to the file points outside the process's allocated address space.

EFBIG	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.
EINTR	The process performing a write received a signal before any data were written, and the signal was set to interrupt the system call.
EINVAL	The *stream* is linked below a multiplexor.
	The seek pointer associated with *fd* was negative.
EIO	An I/O error occurred while reading from or writing to the file system.
	The process is in a background process group and is attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned.
ENOSPC	There is no free space remaining on the file system containing the file.
ENXIO	A hangup occurred on the *stream* being written to.
EPIPE	An attempt is made to write to a pipe that is not open for reading by any process (or to a socket of type **SOCK_STREAM** that is connected to a peer socket.) Note: an attempted write of this kind also causes you to receive a **SIGPIPE** signal from the kernel. If you've not made a special provision to catch or ignore this signal, then your process dies.
ERANGE	*fd* refers to a *stream*, the total number of bytes to be written is outside the minimum and maximum write range, and the minimum value is non-zero.
EWOULDBLOCK	The file was marked for non-blocking I/O, and no data could be written immediately.

In addition to the above, **writev**( ) may set **errno** to:

EINVAL	*iovcnt* was less than or equal to 0, or greater than 16.
	One of the **iov_len** values in the *iov* array was negative.
	The sum of the **iov_len** values in the *iov* array overflowed a 32-bit integer.

A write to a STREAMS file can fail if an error message has been received at the stream head. In this case, **errno** is set to the value included in the error message.

SYSTEM V ERRORS

write( ) fails and sets **errno** as described above, except:

EAGAIN	The descriptor referred to a *stream*, was marked for non-blocking I/O, and no data could be written immediately.
	The **O_NONBLOCK** flag is set for the file descriptor and **write**( ) would block.

SEE ALSO

dup(2V), **fcntl**(2V), **intro**(2), **ioctl**(2), **lseek**(2V), **open**(2V), **pipe**(2V), **select**(2), **sigvec**(2), **signal**(3V)

NAME

intro – introduction to user-level library functions

DESCRIPTION

Section 3 describes user-level library routines. In this release, most user-library routines are listed in alphabetical order regardless of their subsection headings. (This eliminates having to search through several subsections of the manual.) However, due to their special-purpose nature, the routines from the following libraries are broken out into the indicated subsections:

- The Lightweight Processes Library, in subsection 3L.
- The Mathematical Library, in subsection 3M.
- The RPC Services Library, in subsection 3R.

A 3V section number means one or more of the following:

- The man page documents System V behavior only.
- The man page documents default SunOS behavior, and System V behavior as it differs from the default behavior. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents behavior compliant with *IEEE Std 1003.1-1988* (POSIX.1).

The System V Library was formerly documented in a separate manual section. These man pages have been merged into the main portion of section 3. These man pages describe functions that may differ from the default SunOS functions. To use them, compile programs with **/usr/5bin/cc** instead of **/usr/bin/cc**.

Section 3 also documents the library interfaces for *X/Open Portability Guide, Issue 2* (XPG2) compatibility. Where these interfaces differ from the System V versions, the differences are noted. To use the XPG2 compatibility library interfaces, compile programs with **/usr/xpg2bin/cc**.

The libraries provide many different "standard" environments. These environments (including two that are not yet fully supported) are described on **ansic**(7V), **bsd**(7), **posix**(7V), **sunos**(7), **svidii**(7V), **svidiii**(7V), and **xopen**(7V).

The main C library, **/usr/lib/libc.a**, contains many of the functions described in this section, along with entry points for the system calls described in Section 2. This library also includes the Internet networking routines listed under the 3N subsection heading, and routines provided for compatibility with other UNIX operating systems, listed under 3C. Functions associated with the "standard I/O library" are listed under 3S.

User-level routines for access to data structures within the kernel and other processes are listed under 3K. To use these functions, compile programs with the **–lkvm** option for the C compiler, **cc**(1V).

Math library functions are listed under 3M. To use them, compile programs with the **–lm** **cc**(1V) option.

Various specialized libraries, the routines they contain, and the compiler options needed to link with them, are listed under 3X.

FILES

/usr/lib/libc.a	C Library (2, 3, 3N and 3C)
/usr/lib/lib*.a	other "standard" C libraries
/usr/lib/lib*.a	special-purpose C libraries
/usr/5bin/cc	

SEE ALSO

cc(1V), **ld**(1), **nm**(1), **intro**(2)

LIST OF LIBRARY FUNCTIONS

Name	Appears on Page	Description
a64l	**a64l**(3)	convert between long integer and base-64 ASCII string
abort	**abort**(3)	generate a fault
abs	**abs**(3)	integer absolute value
addexportent	**exportent**(3)	get exported file system information
addmntent	**getmntent**(3)	get file system descriptor file entry
aiocancel	**aiocancel**(3)	cancel an asynchronous operation
aioread	**aioread**(3)	asynchronous I/O operations
aiowait	**aiowait**(3)	wait for completion of asynchronous I/O operation
aiowrite	**aioread**(3)	asynchronous I/O operations
alarm	**alarm**(3V)	schedule signal after specified time
alloca	**malloc**(3V)	memory allocator
alphasort	**scandir**(3)	scan a directory
arc	**plot**(3X)	graphics interface
asctime	**ctime**(3V)	convert date and time
assert	**assert**(3V)	program verification
atof	**strtod**(3)	convert string to double-precision number
atoi	**strtol**(3)	convert string to integer
atol	**strtol**(3)	convert string to integer
audit_args	**audit_args**(3)	produce text audit message
audit_text	**audit_args**(3)	produce text audit message
auth_destroy	**rpc_clnt_auth**(3N)	library routines for client side RPC authentication
authdes_create	**secure_rpc**(3N)	library routines for secure remote procedure calls
authdes_getucred	**secure_rpc**(3N)	library routines for secure remote procedure calls
authnone_create	**rpc_clnt_auth**(3N)	library routines for client side RPC authentication
authunix_create	**rpc_clnt_auth**(3N)	library routines for client side RPC authentication
authunix_create_default	**rpc_clnt_auth**(3N)	library routines for client side RPC authentication
bcmp	**bstring**(3)	bit and byte string operations
bcopy	**bstring**(3)	bit and byte string operations
bindresvport	**bindresvport**(3N)	bind a socket to a privileged IP port
bsearch	**bsearch**(3)	binary search a sorted table
bstring	**bstring**(3)	bit and byte string operations
byteorder	**byteorder**(3N)	convert values between host and network byte order
bzero	**bstring**(3)	bit and byte string operations
calloc	**malloc**(3V)	memory allocator
callrpc	**rpc_clnt_calls**(3N)	library routines for client side calls
catclose	**catopen**(3C)	open/close a message catalog
catgetmsg	**catgets**(3C)	get message from a message catalog
catgets	**catgets**(3C)	get message from a message catalog
catopen	**catopen**(3C)	open/close a message catalog
cbc_crypt	**des_crypt**(3)	fast DES encryption
cfgetispeed	**termios**(3V)	terminal control functions
cfgetospeed	**termios**(3V)	terminal control functions
cfree	**malloc**(3V)	memory allocator
cfsetispeed	**termios**(3V)	terminal control functions
cfsetospeed	**termios**(3V)	terminal control functions
circle	**plot**(3X)	graphics interface
clearerr	**ferror**(3V)	stream status inquiries
clnt_broadcast	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_call	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_control	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles

clnt_create	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnt_create_vers	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnt_destroy	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnt_freeres	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_geterr	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_pcreateerror	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnt_perrno	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_perror	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_spcreateerror	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnt_sperrno	**rpc_clnt_calls**(3N)	library routines for client side calls
clnt_sperror	**rpc_clnt_calls**(3N)	library routines for client side calls
clntraw_create	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clnttcp_create	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clntudp_bufcreate	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handles
clock	**clock**(3C)	report CPU time used
closedir	**directory**(3V)	directory operations
closelog	**syslog**(3)	control system log
closepl	**plot**(3X)	graphics interface
cont	**plot**(3X)	graphics interface
conv	**ctype**(3V)	character classification and conversion macros and functions
crypt	**crypt**(3)	password and data encryption
ctermid	**ctermid**(3V)	generate filename for terminal
ctime	**ctime**(3V)	convert date and time
ctype	**ctype**(3V)	character classification and conversion macros and functions
curses	**curses**(3V)	System V terminal screen handling and optimization package
cuserid	**cuserid**(3V)	get character login name of the user
dbm	**dbm**(3X)	data base subroutines
dbm_clearerr	**ndbm**(3)	data base subroutines
dbm_close	**ndbm**(3)	data base subroutines
dbm_delete	**ndbm**(3)	data base subroutines
dbm_error	**ndbm**(3)	data base subroutines
dbm_fetch	**ndbm**(3)	data base subroutines
dbm_firstkey	**ndbm**(3)	data base subroutines
dbm_nextkey	**ndbm**(3)	data base subroutines
dbm_open	**ndbm**(3)	data base subroutines
dbm_store	**ndbm**(3)	data base subroutines
dbmclose	**dbm**(3X)	data base subroutines
dbminit	**dbm**(3X)	data base subroutines
decimal_to_double	**decimal_to_floating**(3)	convert decimal record to floating-point value
decimal_to_extended	**decimal_to_floating**(3)	convert decimal record to floating-point value
decimal_to_single	**decimal_to_floating**(3)	convert decimal record to floating-point value
delete	**dbm**(3X)	data base subroutines
des_crypt	**des_crypt**(3)	fast DES encryption
des_setparity	**des_crypt**(3)	fast DES encryption
directory	**directory**(3V)	directory operations
dlclose	**dlopen**(3X)	simple programmatic interface to the dynamic linker
dlerror	**dlopen**(3X)	simple programmatic interface to the dynamic linker
dlopen	**dlopen**(3X)	simple programmatic interface to the dynamic linker
dlsym	**dlopen**(3X)	simple programmatic interface to the dynamic linker
dn_comp	**resolver**(3)	resolver routines
dn_expand	**resolver**(3)	resolver routines
double_to_decimal	**floating_to_decimal**(3)	convert floating-point value to decimal record
drand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers

dysize	**ctime**(3V)	convert date and time
ecb_crypt	**des_crypt**(3)	fast DES encryption
econvert	**econvert**(3)	output conversion
ecvt	**econvert**(3)	output conversion
edata	**end**(3)	last locations in program
encrypt	**crypt**(3)	password and data encryption
end	**end**(3)	last locations in program
endac	**getacinfo**(3)	get audit control file information
endexportent	**exportent**(3)	get exported file system information
endfsent	**getfsent**(3)	get file system descriptor file entry
endgraent	**getgraent**(3)	get group adjunct file entry
endgrent	**getgrent**(3V)	get group file entry
endhostent	**gethostent**(3N)	get network host entry
endmntent	**getmntent**(3)	get file system descriptor file entry
endnetent	**getnetent**(3N)	get network entry
endnetgrent	**getnetgrent**(3N)	get network group entry
endprotoent	**getprotoent**(3N)	get protocol entry
endpwaent	**getpwaent**(3)	get password adjunct file entry
endpwent	**getpwent**(3V)	get password file entry
endrpcent	**getrpcent**(3N)	get RPC entry
endservent	**getservent**(3N)	get service entry
endttyent	**getttyent**(3)	get ttytab file entry
endusershell	**getusershell**(3)	get legal user shells
erand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
erase	**plot**(3X)	graphics interface
errno	**perror**(3)	system error messages
etext	**end**(3)	last locations in program
ether_aton	**ethers**(3N)	Ethernet address mapping operations
ether_hostton	**ethers**(3N)	Ethernet address mapping operations
ether_line	**ethers**(3N)	Ethernet address mapping operations
ether_ntoa	**ethers**(3N)	Ethernet address mapping operations
ether_ntohost	**ethers**(3N)	Ethernet address mapping operations
ethers	**ethers**(3N)	Ethernet address mapping operations
execl	**execl**(3V)	execute a file
execle	**execl**(3V)	execute a file
execlp	**execl**(3V)	execute a file
execv	**execl**(3V)	execute a file
execvp	**execl**(3V)	execute a file
exit	**exit**(3)	terminate a process after performing cleanup
exportent	**exportent**(3)	get exported file system information
extended_to_decimal	**floating_to_decimal**(3)	convert floating-point value to decimal record
fclose	**fclose**(3V)	close or flush a stream
fconvert	**econvert**(3)	output conversion
fcvt	**econvert**(3)	output conversion
fdopen	**fopen**(3V)	open a stream
feof	**ferror**(3V)	stream status inquiries
ferror	**ferror**(3V)	stream status inquiries
fetch	**dbm**(3X)	data base subroutines
fflush	**fclose**(3V)	close or flush a stream
ffs	**bstring**(3)	bit and byte string operations
fgetc	**getc**(3V)	get character or integer from stream
fgetgraent	**getgraent**(3)	get group adjunct file entry
fgetgrent	**getgrent**(3V)	get group file entry

fgetpwaent	**getpwaent**(3)	get password adjunct file entry
fgetpwent	**getpwent**(3V)	get password file entry
fgets	**gets**(3S)	get a string from a stream
file_to_decimal	**string_to_decimal**(3)	parse characters into decimal record
fileno	**ferror**(3V)	stream status inquiries
firstkey	**dbm**(3X)	data base subroutines
floatingpoint	**floatingpoint**(3)	IEEE floating point definitions
fopen	**fopen**(3V)	open a stream
fprintf	**printf**(3V)	formatted output conversion
fputc	**putc**(3S)	put character or word on a stream
fputs	**puts**(3S)	put a string on a stream
fread	**fread**(3S)	buffered binary input/output
free	**malloc**(3V)	memory allocator
freopen	**fopen**(3V)	open a stream
fscanf	**scanf**(3V)	formatted input conversion
fseek	**fseek**(3S)	reposition a stream
ftell	**fseek**(3S)	reposition a stream
ftime	**time**(3V)	get date and time
ftok	**ftok**(3)	standard interprocess communication package
ftw	**ftw**(3)	walk a file tree
func_to_decimal	**string_to_decimal**(3)	parse characters into decimal record
fwrite	**fread**(3S)	buffered binary input/output
gcd	**mp**(3X)	multiple precision integer arithmetic
gconvert	**econvert**(3)	output conversion
gcvt	**econvert**(3)	output conversion
get_myaddress	**secure_rpc**(3N)	library routines for secure remote procedure calls
getacdir	**getacinfo**(3)	get audit control file information
getacflg	**getacinfo**(3)	get audit control file information
getacinfo	**getacinfo**(3)	get audit control file information
getacmin	**getacinfo**(3)	get audit control file information
getauditflagsbin	**getauditflags**(3)	convert audit flag specifications
getauditflagschar	**getauditflags**(3)	convert audit flag specifications
getc	**getc**(3V)	get character or integer from stream
getchar	**getc**(3V)	get character or integer from stream
getcwd	**getcwd**(3V)	get pathname of current working directory
getenv	**getenv**(3V)	return value for environment name
getexportent	**exportent**(3)	get exported file system information
getexportopt	**exportent**(3)	get exported file system information
getfauditflags	**getfauditflags**(3)	generates the process audit state
getfsent	**getfsent**(3)	get file system descriptor file entry
getfsfile	**getfsent**(3)	get file system descriptor file entry
getfsspec	**getfsent**(3)	get file system descriptor file entry
getfstype	**getfsent**(3)	get file system descriptor file entry
getgraent	**getgraent**(3)	get group adjunct file entry
getgranam	**getgraent**(3)	get group adjunct file entry
getgrent	**getgrent**(3V)	get group file entry
getgrgid	**getgrent**(3V)	get group file entry
getgrnam	**getgrent**(3V)	get group file entry
gethostbyaddr	**gethostent**(3N)	get network host entry
gethostbyname	**gethostent**(3N)	get network host entry
gethostent	**gethostent**(3N)	get network host entry
getlogin	**getlogin**(3V)	get login name
getmntent	**getmntent**(3)	get file system descriptor file entry

getnetbyaddr	**getnetent**(3N)	get network entry
getnetbyname	**getnetent**(3N)	get network entry
getnetent	**getnetent**(3N)	get network entry
getnetgrent	**getnetgrent**(3N)	get network group entry
getnetname	**secure_rpc**(3N)	library routines for secure remote procedure calls
getopt	**getopt**(3)	get option letter from argument vector
getpass	**getpass**(3V)	read a password
getprotobyname	**getprotoent**(3N)	get protocol entry
getprotobynumber	**getprotoent**(3N)	get protocol entry
getprotoent	**getprotoent**(3N)	get protocol entry
getpublickey	**publickey**(3R)	get public or secret key
getpw	**getpw**(3)	get name from uid
getpwaent	**getpwaent**(3)	get password adjunct file entry
getpwanam	**getpwaent**(3)	get password adjunct file entry
getpwent	**getpwent**(3V)	get password file entry
getpwnam	**getpwent**(3V)	get password file entry
getpwuid	**getpwent**(3V)	get password file entry
getrpcbyname	**getrpcent**(3N)	get RPC entry
getrpcbynumber	**getrpcent**(3N)	get RPC entry
getrpcent	**getrpcent**(3N)	get RPC entry
gets	**gets**(3S)	get a string from a stream
getsecretkey	**publickey**(3R)	get public or secret key
getservbyname	**getservent**(3N)	get service entry
getservbyport	**getservent**(3N)	get service entry
getservent	**getservent**(3N)	get service entry
getsubopt	**getsubopt**(3)	parse sub options from a string.
gettext	**gettext**(3)	retrieve a message string, get and set text domain
getttyent	**getttyent**(3)	get ttytab file entry
getttynam	**getttyent**(3)	get ttytab file entry
getusershell	**getusershell**(3)	get legal user shells
getw	**getc**(3V)	get character or integer from stream
getwd	**getwd**(3)	get current working directory pathname
gmtime	**ctime**(3V)	convert date and time
grpauth	**pwdauth**(3)	password authentication routines
gsignal	**ssignal**(3)	software signals
gtty	**stty**(3C)	set and get terminal state
hasmntopt	**getmntent**(3)	get file system descriptor file entry
hcreate	**hsearch**(3)	manage hash search tables
hdestroy	**hsearch**(3)	manage hash search tables
host2netname	**secure_rpc**(3N)	library routines for secure remote procedure calls
hsearch	**hsearch**(3)	manage hash search tables
htonl	**byteorder**(3N)	convert values between host and network byte order
htons	**byteorder**(3N)	convert values between host and network byte order
index	**string**(3)	string operations
inet	**inet**(3N)	Internet address manipulation
inet_addr	**inet**(3N)	Internet address manipulation
inet_lnaof	**inet**(3N)	Internet address manipulation
inet_makeaddr	**inet**(3N)	Internet address manipulation
inet_netof	**inet**(3N)	Internet address manipulation
inet_network	**inet**(3N)	Internet address manipulation
inet_ntoa	**inet**(3N)	Internet address manipulation
initgroups	**initgroups**(3)	initialize supplementary group IDs
initstate	**random**(3)	better random number generator

innetgr	**getnetgrent**(3N)	get network group entry
insque	**insque**(3)	insert/remove element from a queue
isalnum	**ctype**(3V)	character classification and conversion macros and functions
isalpha	**ctype**(3V)	character classification and conversion macros and functions
isascii	**ctype**(3V)	character classification and conversion macros and functions
isatty	**ttyname**(3V)	find name of a terminal
iscntrl	**ctype**(3V)	character classification and conversion macros and functions
isdigit	**ctype**(3V)	character classification and conversion macros and functions
isgraph	**ctype**(3V)	character classification and conversion macros and functions
islower	**ctype**(3V)	character classification and conversion macros and functions
isprint	**ctype**(3V)	character classification and conversion macros and functions
ispunct	**ctype**(3V)	character classification and conversion macros and functions
issecure	**issecure**(3)	indicates whether system is running secure
isspace	**ctype**(3V)	character classification and conversion macros and functions
isupper	**ctype**(3V)	character classification and conversion macros and functions
isxdigit	**ctype**(3V)	character classification and conversion macros and functions
itom	**mp**(3X)	multiple precision integer arithmetic
jrand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
key_decryptsession	**secure_rpc**(3N)	library routines for secure remote procedure calls
key_encryptsession	**secure_rpc**(3N)	library routines for secure remote procedure calls
key_gendes	**secure_rpc**(3N)	library routines for secure remote procedure calls
key_setsecret	**secure_rpc**(3N)	library routines for secure remote procedure calls
kvm_close	**kvm_open**(3K)	specify a kernel to examine
kvm_getcmd	**kvm_getu**(3K)	get the u-area or invocation arguments for a process
kvm_getproc	**kvm_nextproc**(3K)	read system process structures
kvm_getu	**kvm_getu**(3K)	get the u-area or invocation arguments for a process
kvm_nextproc	**kvm_nextproc**(3K)	read system process structures
kvm_nlist	**kvm_nlist**(3K)	get entries from kernel symbol table
kvm_open	**kvm_open**(3K)	specify a kernel to examine
kvm_read	**kvm_read**(3K)	copy data to or from a kernel image or running system
kvm_setproc	**kvm_nextproc**(3K)	read system process structures
kvm_write	**kvm_read**(3K)	copy data to or from a kernel image or running system
l3tol	**l3tol**(3C)	convert between 3-byte integers and long integers
l64a	**a64l**(3)	convert between long integer and base-64 ASCII string
label	**plot**(3X)	graphics interface
lcong48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
ldaclose	**ldclose**(3X)	close a COFF file
ldahread	**ldahread**(3X)	read the archive header of a member of a COFF archive file
ldaopen	**ldopen**(3X)	open a COFF file for reading
ldclose	**ldclose**(3X)	close a COFF file
ldfcn	**ldfcn**(3)	common object file access routines
ldfhread	**ldfhread**(3X)	read the file header of a COFF file
ldgetname	**ldgetname**(3X)	retrieve symbol name for COFF file symbol table entry
ldlinit	**ldlread**(3X)	manipulate line number entries of a COFF file function
ldlitem	**ldlread**(3X)	manipulate line number entries of a COFF file function
ldlread	**ldlread**(3X)	manipulate line number entries of a COFF file function
ldlseek	**ldlseek**(3X)	seek to line number entries of a section of a COFF file
ldnlseek	**ldlseek**(3X)	seek to line number entries of a section of a COFF file
ldnrseek	**ldrseek**(3X)	seek to relocation entries of a section of a COFF file
ldnshread	**ldshread**(3X)	read an indexed/named section header of a COFF file
ldnsseek	**ldsseek**(3X)	seek to an indexed/named section of a COFF file
ldohseek	**ldohseek**(3X)	seek to the optional file header of a COFF file
ldopen	**ldopen**(3X)	open a COFF file for reading

ldrseek	**ldrseek**(3X)	seek to relocation entries of a section of a COFF file
ldshread	**ldshread**(3X)	read an indexed/named section header of a COFF file
ldsseek	**ldsseek**(3X)	seek to an indexed/named section of a COFF file
ldtbindex	**ldtbindex**(3X)	compute the index of a symbol table entry of a COFF file
ldtbread	**ldtbread**(3X)	read an indexed symbol table entry of a COFF file
ldtbseek	**ldtbseek**(3X)	seek to the symbol table of a COFF file
lfind	**lsearch**(3)	linear search and update
line	**plot**(3X)	graphics interface
linemod	**plot**(3X)	graphics interface
localdtconv	**localdtconv**(3)	get date and time formatting conventions
localeconv	**localeconv**(3)	get numeric and monetary formatting conventions
localtime	**ctime**(3V)	convert date and time
lockf	**lockf**(3)	record locking on files
longjmp	**setjmp**(3V)	non-local goto
lrand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
lsearch	**lsearch**(3)	linear search and update
ltol3	**l3tol**(3C)	convert between 3-byte integers and long integers
madd	**mp**(3X)	multiple precision integer arithmetic
madvise	**madvise**(3)	provide advice to VM system
malloc	**malloc**(3V)	memory allocator
malloc_debug	**malloc**(3V)	memory allocator
malloc_verify	**malloc**(3V)	memory allocator
mallocmap	**malloc**(3V)	memory allocator
mblen	**mblen**(3)	multibyte character handling
mbstowcs	**mblen**(3)	multibyte character handling
mbtowc	**mblen**(3)	multibyte character handling
mcmp	**mp**(3X)	multiple precision integer arithmetic
mdiv	**mp**(3X)	multiple precision integer arithmetic
memalign	**malloc**(3V)	memory allocator
memccpy	**memory**(3)	memory operations
memchr	**memory**(3)	memory operations
memcmp	**memory**(3)	memory operations
memcpy	**memory**(3)	memory operations
memory	**memory**(3)	memory operations
memset	**memory**(3)	memory operations
mfree	**mp**(3X)	multiple precision integer arithmetic
min	**mp**(3X)	multiple precision integer arithmetic
mkstemp	**mktemp**(3)	make a unique file name
mktemp	**mktemp**(3)	make a unique file name
mlock	**mlock**(3)	lock (or unlock) pages in memory
mlockall	**mlockall**(3)	lock (or unlock) address space
moncontrol	**monitor**(3)	prepare execution profile
monitor	**monitor**(3)	prepare execution profile
monstartup	**monitor**(3)	prepare execution profile
mout	**mp**(3X)	multiple precision integer arithmetic
move	**plot**(3X)	graphics interface
mp	**mp**(3X)	multiple precision integer arithmetic
mrand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
msub	**mp**(3X)	multiple precision integer arithmetic
msync	**msync**(3)	synchronize memory with physical storage
mtox	**mp**(3X)	multiple precision integer arithmetic
mult	**mp**(3X)	multiple precision integer arithmetic
munlock	**mlock**(3)	lock (or unlock) pages in memory

munlockall	**mlockall**(3)	lock (or unlock) address space
ndbm	**ndbm**(3)	data base subroutines
netname2host	**secure_rpc**(3N)	library routines for secure remote procedure calls
netname2user	**secure_rpc**(3N)	library routines for secure remote procedure calls
nextkey	**dbm**(3X)	data base subroutines
nice	**nice**(3V)	change nice value of a process
nl_init	**setlocale**(3V)	set international environment
nl_langinfo	**nl_langinfo**(3C)	language information
nlist	**nlist**(3V)	get entries from symbol table
nrand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
ntohl	**byteorder**(3N)	convert values between host and network byte order
ntohs	**byteorder**(3N)	convert values between host and network byte order
on_exit	**on_exit**(3)	name termination handler
opendir	**directory**(3V)	directory operations
openlog	**syslog**(3)	control system log
openpl	**plot**(3X)	graphics interface
optarg	**getopt**(3)	get option letter from argument vector
optind	**getopt**(3)	get option letter from argument vector
passwd2des	**xcrypt**(3R)	hex encryption and utility routines
pause	**pause**(3V)	stop until signal
pclose	**popen**(3S)	open or close a pipe (for I/O) from or to a process
perror	**perror**(3)	system error messages
plock	**plock**(3)	lock process, text, or data segment in memory
plot	**plot**(3X)	graphics interface
point	**plot**(3X)	graphics interface
popen	**popen**(3S)	open or close a pipe (for I/O) from or to a process
pow	**mp**(3X)	multiple precision integer arithmetic
printf	**printf**(3V)	formatted output conversion
prof	**prof**(3)	profile within a function
psignal	**psignal**(3)	system signal messages
publickey	**publickey**(3R)	get public or secret key
putc	**putc**(3S)	put character or word on a stream
putchar	**putc**(3S)	put character or word on a stream
putenv	**putenv**(3)	change or add value to environment
putpwent	**putpwent**(3)	write password file entry
puts	**puts**(3S)	put a string on a stream
putw	**putc**(3S)	put character or word on a stream
pwdauth	**pwdauth**(3)	password authentication routines
qsort	**qsort**(3)	quicker sort
rand	**rand**(3V)	simple random number generator
random	**random**(3)	better random number generator
rcmd	**rcmd**(3N)	routines for returning a stream to a remote command
re_comp	**regex**(3)	regular expression handler
re_exec	**regex**(3)	regular expression handler
readdir	**directory**(3V)	directory operations
realloc	**malloc**(3V)	memory allocator
realpath	**realpath**(3)	return the canonicalized absolute pathname
regex	**regex**(3)	regular expression handler
regexp	**regexp**(3)	regular expression compile and match routines
registerrpc	**rpc_svc_calls**(3N)	library routines for registerring servers
remexportent	**exportent**(3)	get exported file system information
remque	**insque**(3)	insert/remove element from a queue
res_init	**resolver**(3)	resolver routines

res_mkquery	**resolver**(3)	resolver routines
res_send	**resolver**(3)	resolver routines
resolver	**resolver**(3)	resolver routines
rewind	**fseek**(3S)	reposition a stream
rewinddir	**directory**(3V)	directory operations
rexec	**rexec**(3N)	return stream to a remote command
rindex	**string**(3)	string operations
rpc	**rpc**(3N)	library routines for remote procedure calls
rpc_createrr	**rpc_clnt_create**(3N)	library routines creating and manipulating CLIENT handle:
rpow	**mp**(3X)	multiple precision integer arithmetic
rresvport	**rcmd**(3N)	routines for returning a stream to a remote command
rtime	**rtime**(3N)	get remote time
ruserok	**rcmd**(3N)	routines for returning a stream to a remote command
scandir	**scandir**(3)	scan a directory
scanf	**scanf**(3V)	formatted input conversion
seconvert	**econvert**(3)	output conversion
seed48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
seekdir	**directory**(3V)	directory operations
setac	**getacinfo**(3)	get audit control file information
setbuf	**setbuf**(3V)	assign buffering to a stream
setbuffer	**setbuf**(3V)	assign buffering to a stream
setegid	**setuid**(3V)	set user and group ID
seteuid	**setuid**(3V)	set user and group ID
setexportent	**exportent**(3)	get exported file system information
setfsent	**getfsent**(3)	get file system descriptor file entry
setgid	**setuid**(3V)	set user and group ID
setgraent	**getgraent**(3)	get group adjunct file entry
setgrent	**getgrent**(3V)	get group file entry
sethostent	**gethostent**(3N)	get network host entry
setjmp	**setjmp**(3V)	non-local goto
setkey	**crypt**(3)	password and data encryption
setlinebuf	**setbuf**(3V)	assign buffering to a stream
setlocale	**setlocale**(3V)	set international environment
setlogmask	**syslog**(3)	control system log
setmntent	**getmntent**(3)	get file system descriptor file entry
setnetent	**getnetent**(3N)	get network entry
setnetgrent	**getnetgrent**(3N)	get network group entry
setprotoent	**getprotoent**(3N)	get protocol entry
setpwaent	**getpwaent**(3)	get password adjunct file entry
setpwent	**getpwent**(3V)	get password file entry
setpwfile	**getpwent**(3V)	get password file entry
setrgid	**setuid**(3V)	set user and group ID
setrpcent	**getrpcent**(3N)	get RPC entry
setruid	**setuid**(3V)	set user and group ID
setservent	**getservent**(3N)	get service entry
setstate	**random**(3)	better random number generator
setttyent	**getttyent**(3)	get ttytab file entry
setuid	**setuid**(3V)	set user and group ID
setusershell	**getusershell**(3)	get legal user shells
setvbuf	**setbuf**(3V)	assign buffering to a stream
sfconvert	**econvert**(3)	output conversion
sgconvert	**econvert**(3)	output conversion
sigaction	**sigaction**(3V)	examine and change signal action

sigaddset	**sigsetops**(3V)	manipulate signal sets
sigdelset	**sigsetops**(3V)	manipulate signal sets
sigemptyset	**sigsetops**(3V)	manipulate signal sets
sigfillset	**sigsetops**(3V)	manipulate signal sets
sigfpe	**sigfpe**(3)	signal handling for specific SIGFPE codes
siginterrupt	**siginterrupt**(3V)	allow signals to interrupt system calls
sigismember	**sigsetops**(3V)	manipulate signal sets
siglongjmp	**setjmp**(3V)	non-local goto
signal	**signal**(3V)	simplified software signal facilities
sigsetjmp	**setjmp**(3V)	non-local goto
sigsetops	**sigsetops**(3V)	manipulate signal sets
single_to_decimal	**floating_to_decimal**(3)	convert floating-point value to decimal record
sleep	**sleep**(3V)	suspend execution for interval
space	**plot**(3X)	graphics interface
sprintf	**printf**(3V)	formatted output conversion
srand48	**drand48**(3)	generate uniformly distributed pseudo-random numbers
srand	**rand**(3V)	simple random number generator
srandom	**random**(3)	better random number generator
sscanf	**scanf**(3V)	formatted input conversion
ssignal	**ssignal**(3)	software signals
stdio	**stdio**(3V)	standard buffered input/output package
store	**dbm**(3X)	data base subroutines
strcasecmp	**string**(3)	string operations
strcat	**string**(3)	string operations
strchr	**string**(3)	string operations
strcmp	**string**(3)	string operations
strcoll	**strcoll**(3)	compare or transform strings using collating information
strcpy	**string**(3)	string operations
strcspn	**string**(3)	string operations
strdup	**string**(3)	string operations
strftime	**ctime**(3V)	convert date and time
string_to_decimal	**string_to_decimal**(3)	parse characters into decimal record
strlen	**string**(3)	string operations
strncasecmp	**string**(3)	string operations
strncat	**string**(3)	string operations
strncmp	**string**(3)	string operations
strncpy	**string**(3)	string operations
strpbrk	**string**(3)	string operations
strptime	**ctime**(3V)	convert date and time
strrchr	**string**(3)	string operations
strspn	**string**(3)	string operations
strstr	**string**(3)	string operations
strtod	**strtod**(3)	convert string to double-precision number
strtok	**string**(3)	string operations
strtol	**strtol**(3)	convert string to integer
strxfrm	**strcoll**(3)	compare or transform strings using collating information
stty	**stty**(3C)	set and get terminal state
svc_destroy	**rpc_svc_create**(3N)	library routines for dealing with the creation of server handles
svc_fds	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_fdset	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_freeargs	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_getargs	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_getcaller	**rpc_svc_reg**(3N)	library routines for RPC servers

svc_getreq	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_getreqset	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_register	**rpc_svc_calls**(3N)	library routines for registerring servers
svc_run	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_sendreply	**rpc_svc_reg**(3N)	library routines for RPC servers
svc_unregister	**rpc_svc_calls**(3N)	library routines for registerring servers
svcerr_auth	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_decode	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_noproc	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_noprog	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_progvers	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_systemerr	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcerr_weakauth	**rpc_svc_err**(3N)	library routines for server side remote procedure call errors
svcfd_create	**rpc_svc_create**(3N)	library routines for dealing with the creation of server handl
svcraw_create	**rpc_svc_create**(3N)	library routines for dealing with the creation of server handl
svctcp_create	**rpc_svc_create**(3N)	library routines for dealing with the creation of server handl
svcudp_bufcreate	**rpc_svc_create**(3N)	library routines for dealing with the creation of server handl
swab	**swab**(3)	swap bytes
sys_siglist	**psignal**(3)	system signal messages
syslog	**syslog**(3)	control system log
system	**system**(3)	issue a shell command
t_accept	**t_accept**(3N)	accept a connect request
t_alloc	**t_alloc**(3N)	allocate a library structure
t_bind	**t_bind**(3N)	bind an address to a transport endpoint
t_close	**t_close**(3N)	close a transport endpoint
t_connect	**t_connect**(3N)	establish a connection with another transport user
t_error	**t_error**(3N)	produce error message
t_free	**t_free**(3N)	free a library structure
t_getinfo	**t_getinfo**(3N)	get protocol-specific service information
t_getstate	**t_getstate**(3N)	get the current state
t_listen	**t_listen**(3N)	listen for a connect request
t_look	**t_look**(3N)	look at the current event on a transport endpoint
t_open	**t_open**(3N)	establish a transport endpoint
t_optmgmt	**t_optmgmt**(3N)	manage options for a transport endpoint
t_rcv	**t_rcv**(3N)	receive normal or expedited data sent over a connection
t_rcvconnect	**t_rcvconnect**(3N)	receive the confirmation from a connect request
t_rcvdis	**t_rcvdis**(3N)	retrieve information from disconnect
t_rcvrel	**t_rcvrel**(3N)	acknowledge receipt of an orderly release indication
t_rcvudata	**t_rcvudata**(3N)	receive a data unit
t_rcvuderr	**t_rcvuderr**(3N)	receive a unit data error indication
t_snd	**t_snd**(3N)	send normal or expedited data over a connection
t_snddis	**t_snddis**(3N)	send user-initiated disconnect request
t_sndrel	**t_sndrel**(3N)	initiate an orderly release
t_sndudata	**t_sndudata**(3N)	send a data unit
t_sync	**t_sync**(3N)	synchronize transport library
t_unbind	**t_unbind**(3N)	disable a transport endpoint
tcdrain	**termios**(3V)	terminal control functions
tcflow	**termios**(3V)	terminal control functions
tcflush	**termios**(3V)	terminal control functions
tcgetattr	**termios**(3V)	terminal control functions
tcgetpgrp	**tcgetpgrp**(3V)	get, set foreground process group ID
tcsendbreak	**termios**(3V)	terminal control functions
tcsetattr	**termios**(3V)	terminal control functions

tcsetpgrp	**tcgetpgrp**(3V)	get, set foreground process group ID
tdelete	**tsearch**(3)	manage binary search trees
telldir	**directory**(3V)	directory operations
tempnam	**tmpnam**(3S)	create a name for a temporary file
termcap	**termcap**(3X)	terminal independent operation routines
termios	**termios**(3V)	terminal control functions
textdomain	**gettext**(3)	retrieve a message string, get and set text domain
tfind	**tsearch**(3)	manage binary search trees
tgetent	**termcap**(3X)	terminal independent operation routines
tgetflag	**termcap**(3X)	terminal independent operation routines
tgetnum	**termcap**(3X)	terminal independent operation routines
tgetstr	**termcap**(3X)	terminal independent operation routines
tgoto	**termcap**(3X)	terminal independent operation routines
time	**time**(3V)	get date and time
timegm	**ctime**(3V)	convert date and time
timelocal	**ctime**(3V)	convert date and time
times	**times**(3V)	get process times
timezone	**timezone**(3C)	get time zone name given offset from GMT
tmpfile	**tmpfile**(3S)	create a temporary file
tmpnam	**tmpnam**(3S)	create a name for a temporary file
toascii	**ctype**(3V)	character classification and conversion macros and functions
tolower	**ctype**(3V)	character classification and conversion macros and functions
toupper	**ctype**(3V)	character classification and conversion macros and functions
tputs	**termcap**(3X)	terminal independent operation routines
tsearch	**tsearch**(3)	manage binary search trees
ttyname	**ttyname**(3V)	find name of a terminal
ttyslot	**ttyslot**(3V)	find the slot in the utmp file of the current process
twalk	**tsearch**(3)	manage binary search trees
tzset	**ctime**(3V)	convert date and time
tzsetwall	**ctime**(3V)	convert date and time
ualarm	**ualarm**(3)	schedule signal after interval in microseconds
ulimit	**ulimit**(3C)	get and set user limits
ungetc	**ungetc**(3S)	push character back into input stream
user2netname	**secure_rpc**(3N)	library routines for secure remote procedure calls
usleep	**usleep**(3)	suspend execution for interval in microseconds
utime	**utime**(3V)	set file times
valloc	**malloc**(3V)	memory allocator
values	**values**(3)	machine-dependent values
varargs	**varargs**(3)	handle variable argument list
vfprintf	**vprintf**(3V)	print formatted output of a varargs argument list
vlimit	**vlimit**(3C)	control maximum system resource consumption
vprintf	**vprintf**(3V)	print formatted output of a varargs argument list
vsprintf	**vprintf**(3V)	print formatted output of a varargs argument list
vsyslog	**vsyslog**(3)	log message with a varargs argument list
vtimes	**vtimes**(3C)	get information about resource utilization
wcstombs	**mblen**(3)	multibyte character handling
wctomb	**mblen**(3)	multibyte character handling
xcrypt	**xcrypt**(3R)	hex encryption and utility routines
xdecrypt	**xcrypt**(3R)	hex encryption and utility routines
xdr	**xdr**(3N)	library routines for external data representation
xdr_accepted_reply	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_array	**xdr_complex**(3N)	library routines for translating complex data types
xdr_authunix_parms	**rpc_xdr**(3N)	XDR library routines for remote procedure calls

xdr_bool	**xdr_simple**(3N)	library routines for translating simple data types
xdr_bytes	**xdr_complex**(3N)	library routines for translating complex data types
xdr_callhdr	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_callmsg	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_char	**xdr_simple**(3N)	library routines for translating simple data types
xdr_destroy	**xdr_create**(3N)	library routines for XDR stream creation
xdr_double	**xdr_simple**(3N)	library routines for translating simple data types
xdr_enum	**xdr_simple**(3N)	library routines for translating simple data types
xdr_float	**xdr_simple**(3N)	library routines for translating simple data types
xdr_free	**xdr_simple**(3N)	library routines for translating simple data types
xdr_getpos	**xdr_admin**(3N)	library routines for management of the XDR stream
xdr_inline	**xdr_admin**(3N)	library routines for management of the XDR stream
xdr_int	**xdr_simple**(3N)	library routines for translating simple data types
xdr_long	**xdr_simple**(3N)	library routines for translating simple data types
xdr_opaque	**xdr_complex**(3N)	library routines for translating complex data types
xdr_opaque_auth	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_pamp	**portmap**(3N)	library routines for RPC bind service
xdr_pmaplist	**portmap**(3N)	library routines for RPC bind service
xdr_pointer	**xdr_complex**(3N)	library routines for translating complex data types
xdr_reference	**xdr_complex**(3N)	library routines for translating complex data types
xdr_rejected_reply	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_replymsg	**rpc_xdr**(3N)	XDR library routines for remote procedure calls
xdr_setpos	**xdr_admin**(3N)	library routines for management of the XDR stream
xdr_short	**xdr_simple**(3N)	library routines for translating simple data types
xdr_string	**xdr_complex**(3N)	library routines for translating complex data types
xdr_u_char	**xdr_simple**(3N)	library routines for translating simple data types
xdr_u_int	**xdr_simple**(3N)	library routines for translating simple data types
xdr_u_long	**xdr_simple**(3N)	library routines for translating simple data types
xdr_u_short	**xdr_simple**(3N)	library routines for translating simple data types
xdr_union	**xdr_complex**(3N)	library routines for translating complex data types
xdr_vector	**xdr_complex**(3N)	library routines for translating complex data types
xdr_void	**xdr_simple**(3N)	library routines for translating simple data types
xdr_wrapstring	**xdr_complex**(3N)	library routines for translating complex data types
xdrmem_create	**xdr_create**(3N)	library routines for XDR stream creation
xdrrec_create	**xdr_create**(3N)	library routines for XDR stream creation
xdrrec_endofrecord	**xdr_admin**(3N)	library routines for management of the XDR stream
xdrrec_eof	**xdr_admin**(3N)	library routines for management of the XDR stream
xdrrec_readbytes	**xdr_admin**(3N)	library routines for management of the XDR stream
xdrrec_skiprecord	**xdr_admin**(3N)	library routines for management of the XDR stream
xdrstdio_create	**xdr_create**(3N)	library routines for XDR stream creation
xencrypt	**xcrypt**(3R)	hex encryption and utility routines
xprt_register	**rpc_svc_calls**(3N)	library routines for registerring servers
xprt_unregister	**rpc_svc_calls**(3N)	library routines for registerring servers
xtom	**mp**(3X)	multiple precision integer arithmetic
yp_all	**ypclnt**(3N)	NIS client interface
yp_bind	**ypclnt**(3N)	NIS client interface
yp_first	**ypclnt**(3N)	NIS client interface
yp_get_default_domain	**ypclnt**(3N)	NIS client interface
yp_master	**ypclnt**(3N)	NIS client interface
yp_match	**ypclnt**(3N)	NIS client interface
yp_next	**ypclnt**(3N)	NIS client interface
yp_order	**ypclnt**(3N)	NIS client interface
yp_unbind	**ypclnt**(3N)	NIS client interface

yp_update	**ypupdate**(3N)	changes NIS information
ypclnt	**ypclnt**(3N)	NIS client interface
yperr_string	**ypclnt**(3N)	NIS client interface
ypprot_err	**ypclnt**(3N)	NIS client interface

NAME

a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS

long a64l(s)
char *s;

char *l64a(l)
long l;

DESCRIPTION

These functions are used to maintain numbers stored in *base-64* ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are '**.**' for 0, '/' for 1, **0** through **9** for 2–11, **A** through **Z** for 12–37, and **a** through z for 38–63.

a64l() takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. If the string pointed to by *s* contains more than six characters, **a64l()** will use the first six.

l64a() takes a **long** argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, **l64a()** returns a pointer to a null string.

BUGS

The value returned by **l64a()** is a pointer into a static buffer, the contents of which are overwritten by each call.

NAME

abort – generate a fault

SYNOPSIS

abort()

DESCRIPTION

abort() first closes all open files if possible, then sends an IOT signal to the process. This signal usually results in termination with a core dump, which may be used for debugging.

It is possible for **abort()** to return control if **SIGIOT** is caught or ignored, in which case the value returned is that of the **kill**(2V) system call.

SEE ALSO

adb(1), **exit**(2V), **kill**(2V), **signal**(3V)

DIAGNOSTICS

If **SIGIOT** is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message '**abort – core dumped**' is written by the shell.

NAME

abs – integer absolute value

SYNOPSIS

abs(i)
int i;

DESCRIPTION

abs( ) returns the absolute value of its integer operand.

SEE ALSO

ieee_functions(3M) for **fabs( )**

BUGS

Applying the **abs( )** function to the most negative integer generates a result which is the most negative integer. That is, **abs(0x80000000)** returns **0x80000000** as a result.

NAME

aiocancel – cancel an asynchronous operation

SYNOPSIS

#include <sys/asynch.h>

int aiocancel(resultp)
aio_result_t *resultp;

DESCRIPTION

aiocancel( ) cancels the asynchronous operation associated with the result buffer pointed to by *resultp*. It may not be possible to immediately cancel an operation which is in progress and in this case, **aiocancel( )** will not wait to cancel it.

Upon successful completion, **aiocancel( )** will return 0 and the requested operation will be canceled. The application will not receive the **SIGIO** completion signal for an asynchronous operation which is successfully canceled.

RETURN VALUES

aiocancel( ) returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

aiocancel( ) will fail if any of the following are true:

EACCES The parameter *resultp* does not correspond to an outstanding asynchronous operation.

The operation could not be cancelled.

EFAULT The parameter *resultp* points to an address that is outside of the address space of the requesting process.

SEE ALSO

aioread(3), **aiowait**(3)

NAME

aioread, aiowrite – asynchronous I/O operations

SYNOPSIS

```
#include <sys/asynch.h>

int aioread(fd, bufp, bufs, offset, whence, resultp)
int fd;
char *bufp;
int bufs;
int offset;
int whence;
aio_result_t *resultp;

int aiowrite(fd, bufp, bufs, offset, whence, resultp)
int fd;
char *bufp;
int bufs;
int offset;
int whence;
aio_result_t *resultp;
```

DESCRIPTION

aioread( ) initiates one asynchronous **read**(2V) and returns control to the calling program. The **read( )** continues concurrently with other activity of the process. An attempt is made to read *bufs* bytes of data from the object referenced by the descriptor *fd* into the buffer pointed to by *bufp*.

aiowrite( ) initiates one asynchronous **write**(2V) and returns control to the calling program. The **write( )** continues concurrently with other activity of the process. An attempt is made to write *bufs* bytes of data from the buffer pointed to by *bufp* to the object referenced by the descriptor *fd*.

On objects capable of seeking, the I/O operation starts at the position specified by *whence* and *offset*. These parameters have the same meaning as the corresponding parameters to the **lseek**(2V) function. On objects not capable of seeking the I/O operation always start from the current position and the parameters *whence* and *offset* are ignored. The seek pointer for objects capable of seeking is not updated by **aioread( )** or **aiowrite( )**. Sequential asynchronous operations on these devices must be managed by the application using the *whence* and *offset* parameters.

The result of the asynchronous operation is stored in the structure pointed to by *resultp*:

```
int aio_return;        /* return value of read( ) or write( ) */
int aio_errno;         /* value of errno for read( ) or write( ) */
```

Upon completion of the operation both *aio_return* and *aio_errno* are set to reflect the result of the operation. AIO_INPROGRESS is not a value used by the system so the client may detect a change in state by initializing *aio_return* to this value.

Notification of the completion of an asynchronous I/O operation may be obtained synchronously through the **aiowait**(3) function, or asynchronously through the signal mechanism. Asynchronous notification is accomplished by generating the **SIGIO** signal. The delivery of this instance of the **SIGIO** signal is reliable in that a signal delivered while the handler is executing is not lost. If the client ensures that **aiowait**(3) returns nothing (using a polling timeout) before returning from the signal handler, no asynchronous I/O notifications are lost. The **aiowait**(3) function is the only way to dequeue an asynchronous notification. Note: **SIGIO** may have several meanings simultaneously: for example, that a descriptor generated **SIGIO** and an asynchronous operation completed. Further, issuing an asynchronous request successfully guarantees that space exists to queue the completion notification.

close(2V), **exit**(2V) and **execve**(2V) will block until all pending asynchronous I/O operations can be cancelled by the system.

It is an error to use the same result buffer in more than one outstanding request. These structures may only be reused after the system has completed the operation.

RETURN VALUES

aioread( ) and **aiowrite( )** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

EBADF *fd* is not a valid file descriptor open for reading.

EFAULT At least one of *bufp* or *resultp* points to an address out side the address space of the requesting process.

EINVAL The parameter *resultp* is currently being used by an outstanding asynchronous request.

EPROCLIM The number of asynchronous requests that the system can handle at any one time has been exceeded

SEE ALSO

close(2V), **execve**(2V), **exit**(2V), **lseek**(2V), **open**(2V), **read**(2V), **sigvec**(2), **write**(2V), **aiocancel**(3), **aiowait**(3)

NAME

aiowait – wait for completion of asynchronous I/O operation

SYNOPSIS

#include <sys/asynch.h>
#include <sys/time.h>

aio_result_t *aiowait(timeout)
struct timeval *timeout;

DESCRIPTION

aiowait() suspends the calling process until one of its outstanding asynchronous I/O operations completes. This provides a synchronous method of notification.

If *timeout* is a non-zero pointer, it specifies a maximum interval to wait for the completion of an asynchronous I/O operation. If *timeout* is a zero pointer, then **aiowait()** blocks indefinitely. To effect a poll, the *timeout* parameter should be non-zero, pointing to a zero-valued *timeval* structure. The *timeval* structure is defined in **<sys/time.h>** as:

```
struct timeval {
        long  tv_sec;          /* seconds           */
        long  tv_usec;         /* and microseconds  */
};
```

NOTES

aiowait() is the only way to dequeue an asynchronous notification. It may be used either inside a **SIGIO** signal handler or in the main program. Note: one **SIGIO** signal may represent several queued events.

RETURN VALUES

On success, **aiowait()** returns a pointer to the result structure used when the completed asynchronous I/O operation was requested. On failure, it returns –1 and sets **errno** to indicate the error. **aiowait() returns** 0 if the time limit expires.

ERRORS

EFAULT	*timeout* points to an address outside the address space of the requesting process.
EINTR	A signal was delivered before an asynchronous I/O operation completed.
	The time limit expired.
EINVAL	There are no outstanding asynchronous I/O requests.

SEE ALSO

aiocancel(3), **aioread**(3)

NAME

alarm – schedule signal after specified time

SYNOPSIS

unsigned int alarm(seconds)
unsigned int seconds;

DESCRIPTION

alarm( ) sends the signal **SIGALRM** (see **sigvec**(2)), to the invoking process after *seconds* seconds. Unless caught or ignored, the signal terminates the process.

alarm( ) requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any **alarm( )** request is canceled. Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

sigpause(2V), **sigvec**(2), **signal**(3V), **sleep**(3V), **ualarm**(3), **usleep**(3)

WARNINGS

alarm( ) is slightly incompatible with the default version of **sleep(3V).** The alarm signal is not sent when one would expect for programs that wait one second of clock time between successive calls to **sleep( )**. Each **sleep( )** call postpones the alarm signal that would have been sent during the requested sleep period for one second. Use System V **sleep**(3V) to avoid this delay.

NAME

assert – program verification

SYNOPSIS

#include <assert.h>

assert(expression)

DESCRIPTION

assert() is a macro that indicates *expression* is expected to be true at this point in the program. If *expression* is false (0), it displays a diagnostic message on the standard output and exits (see **exit**(2V)). Compiling with the **cc**(1V) option **–DNDEBUG**, or placing the preprocessor control statement

#define NDEBUG

before the "**#include <assert.h>**" statement effectively deletes **assert()** from the program.

SYSTEM V DESCRIPTION

The System V version of **assert()** calls **abort**(3) rather than **exit()**.

SEE ALSO

cc(1V), **exit**(2V), **abort**(3)

DIAGNOSTICS

Assertion failed: file *f* **line** *n*

The expression passed to the **assert()** statement at line *n* of source file *f* was false.

SYSTEM V DIAGNOSTICS

Assertion failed: *expression*, **file** *f*, **line** *n*

The *expression* passed to the **assert()** statement at line *n* of source file *f* was false.

NAME

audit_args, audit_text – produce text audit message

SYNOPSIS

```
#include &lt;sys/label.h&gt;
#include &lt;sys/audit.h&gt;

audit_args(event, argc, argv)
int event;
int argc;
char **argv;

audit_text(event, error, retval, argc, argv)
int event;
int error;
int retval;
int argc;
char **argv;
```

DESCRIPTION

These functions provide text interfaces to the **audit**(2) system call. In both calls, the *event* parameter identifies the event class of the action, and *argc* is the number of strings found in the vector *argv*. The **error** parameter is used to determine the failure or success of the audited operation. A negative value is always audited. A zero value is audited as a successful event. A positive value is audited as an event failure. The *retval* parameter is the return value or exit code that the invoking program will have.

audit_args( ) is equivalent to **audit_text( )** with **error** and *retval* parameters of −1.

SEE ALSO

audit(2)

NAME

bindresvport – bind a socket to a privileged IP port

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

int bindresvport(sd, sin)
int sd;
struct sockaddr_in *sin;

DESCRIPTION

bindresvport() is used to bind a socket descriptor to a privileged IP port, that is, a port number in the range 0-1023. The routine returns 0 if it is successful, otherwise −1 is returned and **errno** set to reflect the cause of the error. This routine differs with **rresvport** (see **rcmd**(3N)) in that this works for any IP socket, whereas **rresvport()** only works for TCP.

Only root can bind to a privileged port; this call will fail for any other users.

SEE ALSO

rcmd(3N)

NAME

bsearch – binary search a sorted table

SYNOPSIS

#include <search.h>

char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)
unsigned nel;
int (*compar)( );

DESCRIPTION

bsearch( ) is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *key* points to a datum instance to be sought in the table. *base* points to the element at the base of the table. *nel* is the number of elements in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero as accordingly the first argument is to be considered less than, equal to, or greater than the second.

EXAMPLE

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node, in which case it prints out the string and its length, or it prints an error message.

```
#include <stdio.h>
#include <search.h>
#define TABSIZE         1000
struct node {                       /* these are stored in the table */
        char *string;
        int length;
};
struct node table[TABSIZE];         /* table to be searched */
        .
        .
        .
{
        struct node *node_ptr, node;
        int node_compare( ); /* routine to compare 2 nodes */
        char str_space[20];  /* space to read string into */
        .
        .
        .
        node.string = str_space;
        while (scanf(" %s", node.string) != EOF) {
                node_ptr = (struct node *)bsearch((char *)(&node),
                           (char *)table, TABSIZE,
                           sizeof(struct node), node_compare);
                if (node_ptr != NULL) {
                        (void)printf("string = %20s, length = %d\n",
                                node_ptr->string, node_ptr->length);
                } else {
                        (void)printf("not found: %s\n", node.string);
                }
        }
}
/*
        This routine compares two nodes based on an
        alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
        return strcmp(node1->string, node2->string);
}
```

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO

hsearch(3), **lsearch**(3), **qsort**(3), **tsearch**(3)

DIAGNOSTICS

A NULL pointer is returned if the key cannot be found in the table.

NAME

bstring, bcopy, bcmp, bzero, ffs – bit and byte string operations

SYNOPSIS

void
bcopy(b1, b2, length)
char *b1, *b2;
int length;

int bcmp(b1, b2, length)
char *b1, *b2;
int length;

void
bzero(b, length)
char *b;
int length;

int ffs(i)
int i;

DESCRIPTION

The functions **bcopy**, **bcmp**, and **bzero( )** operate on variable length strings of bytes. They do not check for null bytes as the routines in **string**(3) do.

bcopy( ) copies *length* bytes from string *b1* to the string *b2*. Overlapping strings are handled correctly.

bcmp( ) compares byte string *b1* against byte string *b2*, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be *length* bytes long. **bcmp( )** of length zero bytes always returns zero.

bzero( ) places *length* 0 bytes in the string *b*.

ffs( ) finds the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1 from the right. A return value of zero indicates that the value passed is zero.

NOTES

The **bcmp( )** and **bcopy( )** routines take parameters backwards from **strcmp( )** and **strcpy( )**.

SEE ALSO

string(3)

NAME

byteorder, htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On Sun-2, Sun-3 and Sun-4 systems, these routines are defined as NULL macros in the include file **<netinet/in.h>**. On Sun386i systems, these routines are functional since its host byte order is different from network byte order.

These routines are most often used in conjunction with Internet addresses and ports as returned by **gethostent**(3N) and **getservent**(3N).

SEE ALSO

gethostent(3N), **getservent**(3N)

NAME

catgets, catgetmsg – get message from a message catalog

SYNOPSIS

```
#include <nl_types.h>

char *catgets(catd, set_num, msg_num, s)
nl_catd catd;
int set_num, msg_num;
char *s;

char *catgetmsg(catd, set_num, msg_num, buf, buflen)
nl_catd catd;
int set_num;
int msg_num;
int buflen;
```

DESCRIPTION

catgets() reads the message *msg_num*, in set *set_num*, from the message catalog identified by *catd*. *catd* is a catalog descriptor returned from an earlier call to **catopen**(3C). *s* points to a default message string which will be returned by **catgets()** if the identified message catalog is not currently available. The message-text is contained in an internal buffer area and should be copied by the application if it is to be saved or re-used after further calls to **catgets()**.

catgetmsg() attempts to read up to *buflen* −1 bytes of a message string into the area pointed to by *buf*. *buflen* is an integer value containing the size in bytes of *buf*. The return string is always terminated with a null byte.

RETURN VALUES

On success, **catgets()** returns a pointer to an internal buffer area containing the null-terminated message string. **catgets()** returns a pointer to *s* if it fails because the message catalog specified by *catd* is not currently available. Otherwise, **catgets()** returns a pointer to an empty string if the message catalog is available but does not contain the specified message.

On success, **catgetmsg()** returns a pointer to the message string in *buf*. If *catd* is invalid or if *set_num* or *msg_num* is not in the message catalog, **catgetmsg()** returns a pointer to an empty string.

SEE ALSO

catopen(3C), **locale**(5)

NAME

catopen, catclose – open/close a message catalog

SYNOPSIS

#include <nl_types.h>

nl_catd catopen(name, oflag)
char *name;
int oflag;

int catclose(catd)
nl_catd catd;

DESCRIPTION

catopen( ) opens a message catalog and returns a catalog descriptor. *name* specifies the name of the message catalog to be opened. If *name* contains a '/' then *name* specifies a pathname for the message catalog. Otherwise, the environment variable **NLSPATH** is used with *name* substituted for **%N** (see **locale**(5)). If **NLSPATH** does not exist in the environment, or if a message catalog cannot be opened in any of the paths specified by **NLSPATH**, the **/etc/locale/LC_MESSAGES/***locale* directory is searched for a message catalog with filename *name*, followed by the **/usr/share/lib/locale/LC_MESSAGES/***locale* directory. In both cases *locale* stands for the current setting of the **LC_MESSAGES** category of locale.

oflag is reserved for future use and should be set to 0 (zero). The results of setting this field to any other value are undefined.

catclose( ) closes the message catalog identified by *catd*. It invalidates any following references to the message catalog defined by *catd*.

RETURN VALUES

catopen( ) returns a message catalog descriptor on success. On failure, it returns –1.

catclose( ) returns:

0 on success.

–1 on failure.

SEE ALSO

catgets(3C), **locale**(5)

NOTES

Using **catopen( )** and **catclose( )** in conjunction with **gettext( )** or **textdomain( )** (see **gettext**(3)) is undefined.

NAME

clock – report CPU time used

SYNOPSIS

long clock ( )

DESCRIPTION

clock() returns the amount of CPU time (in microseconds) used since the first call to **clock**. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed **wait**(2V) or **system**(3).

The resolution of the clock is 16.667 milliseconds.

SEE ALSO

wait(2V), **system**(3), **times**(3V)

BUGS

The value returned by **clock()** is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).

NAME

crypt, _crypt, setkey, encrypt – password and data encryption

SYNOPSIS

char *crypt(key, salt)
char *key, *salt;

char *_crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;

DESCRIPTION

crypt() is the password encryption routine, based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to **crypt()** is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. Unless it starts with '##' or '#$', the *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

If the *salt* string starts with '##', **pwdauth**(3) is called. If *pwdauth* returns TRUE, the salt is returned from **crypt.** Otherwise, NULL is returned. If the *salt* string starts with **'#$'**, **grpauth** (see **pwdauth**(3)) is called. If **grpauth** returns TRUE, the salt is returned from **crypt**. Otherwise, NULL is returned. If there is a valid reason not to have this authentication happen, calling *_crypt* avoids authentication.

The *setkey* and *encrypt* entries provide (rather primitive) access to the DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the above mentioned algorithm to encrypt or decrypt the string *block* with the function *encrypt*.

The argument to the *encrypt* entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is zero, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

login(1), **passwd**(1), **getpass**(3V), **pwdauth**(3), **passwd**(5)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ctermid – generate filename for terminal

SYNOPSIS

#include <stdio.h>
char *ctermid (s)
char *s;

DESCRIPTION

ctermid( ) generates the pathname of the controlling terminal for the current process, and stores it in a string.

If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to **ctermid( )**, and the address of which is returned. Otherwise, *s* is assumed to point to a character array of at least **L_ctermid** elements; the path name is placed in this array and the value of *s* is returned. The constant **L_ctermid** is defined in **<stdio.h>** header file.

ctermid( ) returns a pointer to a null string if it fails, or if the pathname that would refer to the controlling terminal cannot be determined.

SEE ALSO

ttyname(3V)

NOTES

The difference between **ctermid( )** and **ttyname**(3V) is that **ttyname( )** must be passed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while **ctermid( )** returns a string (**/dev/tty**) that will refer to the terminal if used as a file name. Thus **ttyname( )** is useful only if the process already has at least one file open to a terminal. **ctermid( )** is useful largely for making code portable to (non-UNIX) systems where the current terminal is referred to by a name other than **/dev/tty**.

NAME

ctime, asctime, dysize, gmtime, localtime, strftime, strptime, timegm, timelocal, tzset, tzsetwall – convert date and time

SYNOPSIS

```
#include <time.h>

char *ctime(clock)
time_t *clock;

char *asctime(tm)
struct tm *tm;

int dysize(y)
int y;

struct tm *gmtime(clock)
time_t *clock;

struct tm *localtime(clock)
time_t *clock;

int strftime(buf, bufsize, fmt, tm)
char *buf;
int bufsize;
char *fmt;
struct tm *tm;

char *strptime(buf, fmt, tm)
char *buf;
char *fmt;
struct tm *tm;

time_t timegm(tm)
struct tm *tm;

time_t timelocal(tm)
struct tm *tm;

void tzset( )

void tzsetwall( )
```

SYSTEM V SYNOPSIS

In addition to the routines above, the following variables are available:

```
extern long timezone;

extern int daylight;

extern char *tzname[2];
```

DESCRIPTION

ctime( ) converts a long integer, pointed to by *clock*, to a 26-character string of the form produced by **asctime( )**. It first breaks down *clock* to a **tm** structure by calling **localtime( )**, and then calls **asctime( )** to convert that **tm** structure to a string.

asctime( ) converts a time value contained in a **tm** structure to a 26-character string of the form:

Sun Sep 16 01:03:52 1973\n\0

Each field has a constant width. **asctime( )** returns a pointer to the string.

dysize( ) returns the number of days in the argument year, either 365 or 366. **localtime( )** and **gmtime( )** return pointers to structures containing the time, broken down into various components of that time represented in a particular time zone. **localtime( )** breaks down a time specified by the value pointed to by the *clock* argument, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **localtime( )** calls **tzset( )** (if **tzset( )** has not been called in the current process). **gmtime( )** breaks down a time specified by the value pointed to by the *clock* argument into GMT, which is the time the system uses.

strftime( ) converts a time value contained in the **tm** structure pointed to by *tm* to a character string in a format specified by *fmt*. The character string is placed into the array pointed to by *buf*, which is assumed to contain room for at least *buflen* characters. If the result contains no more than *buflen* characters, **strftime( )** returns the number of characters produced (not including the terminating null character). Otherwise, it returns zero and the contents of the array are indeterminate. *fmt* is a character string that consists of field descriptors and text characters, reminiscent of **printf**(3V). Each field descriptor consists of a % character followd by another character that specifies the replacement for the field descriptor. All other characters are copied from *fmt* into the result. The following field descriptors are supported:

%%	same as %
%a	day of week, using locale's abbreviated weekday names
%A	day of week, using locale's full weekday names
%b **%h**	month, using locale's abbreviated month names
%B	month, using locale's full month names
%c	date and time as %x %X
%C	date and time, in locale's long-format date and time representation
%d	day of month (01-31)
%D	date as %m/%d/%y
%e	day of month (1-31; single digits are preceded by a blank)
%H	hour (00-23)
%I	hour (00-12)
%j	day number of year (001-366)
%k	hour (0-23; single digits are preceded by a blank)
%l	hour (1-12; single digits are preceded by a blank)
%m	month number (01-12)
%M	minute (00-59)
%n	same as **\n**
%p	locale's equivalent of AM or PM, whichever is appropriate
%r	time as %I:%M:%S %p
%R	time as %H:%M
%S	seconds (00-59)
%t	same as **\t**
%T	time as %H:%M:%S
%U	week number of year (01-52), Sunday is the first day of the week
%w	day of week; Sunday is day 0
%W	week number of year (01-52), Monday is the first day of the week
%x	date, using locale's date format
%X	time, using locale's time format

%y year within century (00-99)

%Y year, including century (fore example, 1988)

%Z time zone abbreviation

The difference between **%U** and **%W** lies in which day is counted as the first day of the week. Week number 01 is the first week with four or more January days in it.

strptime( ) converts the character string pointed to by *buf* to a time value, which is stored in the **tm** structure pointed to by *tm*, using the format specified by *fmt*. A pointer to the character following the last character in the string pointed to by *buf* is returned. *fmt* is a character string that consists of field descriptors and text characters, reminiscent of **scanf**(3v). Each field descriptor consists of a % character followd by another character that specifies the replacement for the field descriptor. All other characters are copied from *fmt* into the result. The following field descriptors are supported:

%% same as %

%a
%A day of week, using locale's weekday names; either the abbreviated or full name may be specified

%b
%B
%h month, using locale's month names; either the abbreviated or full name may be specified

%c date and time as %x %X

%C date and time, in locale's long-format date and time representation

%d
%e day of month (1-31; leading zeroes are permitted but not required)

%D date as %m/%d/%y

%H
%k hour (0-23; leading zeroes are permitted but not required)

%I
%l hour (0-12; leading zeroes are permitted but not required)

%j day number of year (001-366)

%m month number (1-12; leading zeroes are permitted but not required)

%M minute (0-59; leading zeroes are permitted but not required)

%p locale's equivalent of AM or PM

%r time as %I:%M:%S %p

%R time as %H:%M

%S seconds (0-59; leading zeroes are permitted but not required)

%T time as %H:%M:%S

%x date, using locale's date format

%X time, using locale's time format

%y year within century (0-99; leading zeroes are permitted but not required)

%Y year, including century (for example, 1988)

Case is ignored when matching items such as month or weekday names. The **%M**, **%S**, **%y**, and **%Y** fields are optional; if they would be matched by white space, the match is suppressed and the appropriate field of the **tm** structure pointed to by *tm* is left unchanged. If any of the format items **%d**, **%e**, **%H**, **%k**, **%I**, **%l**, **%m**, **%M**, **%S**, **%y**, or **%Y** are matched, but the string that matches them is followed by white space, all subsequent items in the format string are skipped up to white space or the end of the format. The net result is that, for example, the format **%m/%d/%y** can be matched by the string **12/31**; the **tm_mon** and **tm_mday** fields of the **tm** structure pointed to by *tm* will be set to 11 and 31, respectively, while the **tm_year** field will be unchanged.

timelocal( ) and **timegm( )** convert the time specified by the value pointed to by the *tm* argument to a time value that represents that time expressed as the number of seconds since Jan. 1, 1970, 00:00, Greenwich Mean Time. **timelocal( )** converts a **tm** structure that represents local time, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **timelocal( )** calls **tzset( )** (if **tzset( )** has not been called in the current process). **timegm( )** converts a **tm** structure that represents GMT.

tzset( ) uses the value of the environment variable TZ to set time conversion information used by **localtime( )**. If TZ is absent from the environment, the an available approximation to local wall clock time is used by **localtime( )**. If TZ appears in the environment but its value is a null string, Greenwich Mean Time is used; if TZ appears and begins with a slash, it is used as the absolute pathname of the *tzfile-format* (see **tzfile**(5)) file from which to read the time conversion information; if TZ appears and begins with a character other than a slash, it is used as a pathname relative to a system time conversion information directory.

tzsetwall( ) sets things up so that **localtime( )** returns the best available approximation of local wall clock time.

Declarations of all the functions and externals, and the **tm** structure, are in the **<time.h>** header file. The structure (of type) **tm** structure includes the following fields:

```
int tm_sec;        /* seconds (0 - 59) */
int tm_min;        /* minutes (0 - 59) */
int tm_hour;       /* hours (0 - 23) */
int tm_mday;       /* day of month (1 - 31) */
int tm_mon;        /* month of year (0 - 11) */
int tm_year;       /* year – 1900 */
int tm_wday;       /* day of week (Sunday = 0) */
int tm_yday;       /* day of year (0 - 365) */
int tm_isdst;      /* 1 if DST in effect */
char *tm_zone;     /* abbreviation of timezone name */
long tm_gmtoff;    /* offset from GMT in seconds */
```

tm_isdst is non-zero if Daylight Savings Time is in effect. **tm_zone** points to a string that is the name used for the local time zone at the time being converted. **tm_gmtoff** is the offset (in seconds) of the time represented from GMT, with positive values indicating East of Greenwich.

SYSTEM V DESCRIPTION

The external **long** variable **timezone** contains the difference, in seconds, between GMT and local standard time (in PST, **timezone** is 8*60*60). If this difference is not a constant, **timezone** will contain the value of the offset on January 1, 1970 at 00:00 GMT. Since this is not necessarily the same as the value at some particular time, the time in question should be converted to a **tm** structure using **localtime( )** and the **tm_gmtoff** field of that structure should be used. The external variable **daylight** is non-zero if and only if Daylight Savings Time would be in effect within the current time zone at some time; it does not indicate whether Daylight Savings Time is currently in effect.

The external variable **tzname** is an array of two **char *** pointers. The first pointer points to a character string that is the name of the current time zone when Daylight Savings Time is not in effect; the second one, if Daylight Savings Time conversion should be applied, points to a character string that is the name of the current time zone when Daylight Savings Time is in effect. These strings are updated by **localtime**( ) whenever a time is converted. If Daylight Savings Time is in effect at the time being converted, the second pointer is set to point to the name of the current time zone at that time, otherwise the first pointer is so set.

timezone, **daylight**, and **tzname** are retained for compatibility with existing programs.

FILES

/usr/share/lib/zoneinfo	standard time conversion information directory
/usr/share/lib/zoneinfo/localtime	local time zone file

SEE ALSO

gettimeofday(2), **getenv**(3V), **time**(3V), **environ**(5V), **tzfile**(5)

BUGS

The return values point to static data, whose contents are overwritten by each call. The **tm_zone** field of a returned **tm** structure points to a static array of characters, which will also be overwritten at the next call (and by calls to **tzset**( ) or **tzsetwall**( )).

NAME

ctype, conv, isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii, isgraph, toupper, tolower, toascii – character classification and conversion macros and functions

SYNOPSIS

#include <ctype.h>

isalpha(c)

. . .

DESCRIPTION

Character Classification Macros

These macros classify character-coded integer values according to the rules of the coded character set defined by the character type information in the program's locale (category **LC_CTYPE**). On program startup the **LC_CTYPE** category of locale is equivalent to the **"C"** locale.

In the **"C"** locale, or in a locale where the character type information is not defined, characters are classified according to the rules of the US-ASCII 7-bit coded character set. The control characters are those below 040 (and the single byte 0177) (DEL). See **ascii**(7).

In all cases that argument is an **int**, the value of which must be representable as an **unsigned char** or must equal the value of the macro EOF. If the argument has any other value, the behavior is undefined.

Each is a predicate returning nonzero for true, zero for false. **isascii**( ) is defined on all integer values.

isalpha(*c*)	*c* is a letter.
isupper(*c*)	*c* is an upper case letter.
islower(*c*)	*c* is a lower case letter.
isdigit(*c*)	*c* is a digit [0-9].
isxdigit(*c*)	*c* is a hexadecimal digit [0-9], [A-F], or [a-f].
isalnum(*c*)	*c* is an alphanumeric character, that is, *c* is a letter or a digit.
isspace(*c*)	*c* is a SPACE, TAB, RETURN, NEWLINE, FORMFEED, or vertical tab character.
ispunct(*c*)	*c* is a punctuation character (neither control nor alphanumeric).
isprint(*c*)	*c* is a printing character.
iscntrl(*c*)	*c* is a delete character or ordinary control character.
isascii(*c*)	*c* is an ASCII character, code less than 0200.
isgraph(*c*)	*c* is a visible graphic character.

Character Conversion Macros

toascii(*c*)

Masks *c* with the correct value so that *c* is guaranteed to be an ASCII character in the range 0 through 0x7f. Will not perform mapping from a non-ASCII coded character set into ASCII.

Character Conversion Functions

These functions perform simple conversions on single characters. They replace the previous macro definitions which did not extend to support variant settings of the **LC_CTYPE** locale category.

toupper(*c*) Converts *c* to its upper-case equivalent. This function works correctly for all coded character sets and all characters within such sets selected by a valid setting of the **LC_CTYPE** locale category.

tolower(*c*) Converts *c* to its lower-case equivalent. This function works correctly for all coded character sets and all characters within such sets selected by a valid setting of the **LC_CTYPE** locale category.

If the argument to any of these macros is not in the domain of the function, the result is undefined.

SYSTEM V DESCRIPTION

Character Conversion Macros

The macros **_toupper**() and **_tolower**() are faster than the equivalent functions (**toupper**() and **tolower**()) but only work properly on a restricted range of characters, and will not work on a **LC_CTYPE** category other than the default **"C"** (ASCII).

These macros perform simple conversions on single characters.

_toupper(*c*) converts *c* to its upper-case equivalent. Note: This *only* works where *c* is known to be a lower-case character to start with (presumably checked using **islower**()).

_tolower(*c*) converts *c* to its lower-case equivalent. Note: This *only* works where *c* is known to be a upper-case character to start with (presumably checked using **isupper**()).

SEE ALSO

setlocale(3V), **ascii**(7), **iso_8859_1**(7)

NAME

curses – System V terminal screen handling and optimization package

SYNOPSIS

The **curses** manual page is organized as follows:

In SYNOPSIS

- compiling information
- summary of parameters used by **curses** routines

In SYSTEM V SYNOPSIS:

- compiling information

In DESCRIPTION and SYSTEM V DESCRIPTION:

- An overview of how **curses** routines should be used

In ROUTINES, descriptions of **curses** routines are grouped under the appropriate topics:

- Overall Screen Manipulation
- Window and Pad Manipulation
- Output
- Input
- Output Options Setting
- Input Options Setting
- Environment Queries
- Low-level Curses Access
- Miscellaneous
- Use of **curscr**

In SYSTEM V ROUTINES, descriptions of **curses** routines are grouped under the appropriate topics:

- Overall Screen Manipulation
- Window and Pad Manipulation
- Output
- Input
- Output Options Setting
- Input Options Setting
- Environment Queries
- Soft Labels
- Low-level Curses Access
- Terminfo-Level Manipulations
- Termcap Emulation
- Miscellaneous
- Use of **curscr**

Then come sections on:

- SYSTEM V ATTRIBUTES
- SYSTEM V FUNCTION KEYS

- LINE GRAPHICS

cc [*flags*] *files* **–lcurses –ltermcap** [*libraries*]

#include <curses.h> (automatically includes **<stdio.h>** and **<unctl.h>**.)

The parameters in the following list are not global variables. This is a summary of the parameters used by the **curses** library routines. All routines return the **int** values ERR or OK unless otherwise noted. Routines that return pointers always return NULL on error. ERR, OK , and NULL are all defined in **<curses.h>**.) Routines that return integers are not listed in the parameter list below.

bool bf

char **area,*boolnames[ ], *boolcodes[ ], *boolfnames[ ], *bp
char *cap, *capname, codename[2], erasechar, *filename, *fmt
char *keyname, killchar, *label, *longname
char *name, *numnames[ ], *numcodes[ ], *numfnames[ ]
char *slk_label, *str, *strnames[ ], *strcodes[ ], *strfnames[ ]
char *term, *tgetstr, *tigetstr, *tgoto, *tparm, *type

chtype attrs, ch, horch, vertch

FILE *infd, *outfd

int begin_x, begin_y, begline, bot, c, col, count
int dmaxcol, dmaxrow, dmincol, dminrow, *errret, fildes
int (*init( )), labfmt, labnum, line
int ms, ncols, new, newcol, newrow, nlines, numlines
int oldcol, oldrow, overlay
int p1, p2, p9, pmincol, pminrow, (*putc( )), row
int smaxcol, smaxrow, smincol, sminrow, start
int tenths, top, visibility, x, y

SCREEN *new, *newterm, *set_term

TERMINAL *cur_term, *nterm, *oterm

va_list varglist

WINDOW *curscr, *dstwin, *initscr, *newpad, *newwin, *orig

WINDOW *pad, *srcwin, *stdscr, *subpad, *subwin, *win

SYSTEM V SYNOPSIS

/usr/5bin/cc [*flag* ...] *file* ... **–lcurses** [*library* ...]

#include <curses.h> (automatically includes **<stdio.h>**, **<termio.h>**, and **<unctrl.h>**).

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the **refresh( )** tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine **initscr( )** must be called before any of the other routines that deal with windows and screens are used. The routine **endwin( )** should be called before exiting.

SYSTEM V DESCRIPTION

The **curses** routines give the user a terminal-independent method of updating screens with reasonable optimization.

In order to initialize the routines, the routine **initscr( )** or **newterm( )** must be called before any of the other routines that deal with windows and screens are used. Three exceptions are noted where they apply. The routine **endwin( )** must be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented programs want this) after calling **initscr( )** you should call '**cbreak ( ); noecho ( );**' Most programs would additionally call '**nonl ( ); intrflush(stdscr, FALSE); keypad(stdscr, TRUE);**'.

Before a **curses** program is run, a terminal's TAB stops should be set and its initialization strings, if defined, must be output. This can be done by executing the **tset** command in your **.profile** or **.login** file. For further details, see **tset**(1) and the **Tabs and Initialization** subsection of **terminfo**(5V).

The **curses** library contains routines that manipulate data structures called *windows* that can be thought of as two-dimensional arrays of characters representing all or part of a terminal screen. A default window called **stdscr** is supplied, which is the size of the terminal screen. Others may be created with **newwin( )**. Windows are referred to by variables declared as **WINDOW ***; the type **WINDOW** is defined in **<curses.h>** to be a C structure. These data structures are manipulated with routines described below, among which the most basic are **move( )** and **addch( )**. More general versions of these routines are included with names beginning with **w**, allowing you to specify a window. The routines not beginning with **w** usually affect **stdscr**. Then **refresh( )** is called, telling the routines to make the user's terminal screen look like **stdscr**. The characters in a window are actually of type **chtype**, so that other information about the character may also be stored with each character.

Special windows called *pads* may also be manipulated. These are windows that are not constrained to the size of the screen and whose contents need not be displayed completely. See the description of **newpad( )** under **Window and Pad Manipulation** for more information.

In addition to drawing characters on the screen, video attributes may be included that cause the characters to show up in modes such as underlined or in reverse video on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, **curses** is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in **<curses.h>**, such as **A_REVERSE**, **ACS_HLINE**, and **KEY_LEFT**.

curses also defines the **WINDOW *** variable, **curscr**, which is used only for certain low-level operations like clearing and redrawing a garbaged screen. **curscr** can be used in only a few routines. If the window argument to **clearok( )** is **curscr**, the next call to **wrefresh( )** with any window will clear and repaint the screen from scratch. If the window argument to **wrefresh( )** is **curscr**, the screen in immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" function. More information on using **curscr** is provided where its use is appropriate.

The environment variables **LINES** and **COLUMNS** may be set to override **curses**'s idea of how large a screen is.

If the environment variable **TERMINFO** is defined, any program using **curses** will check for a local terminal definition before checking in the standard place. For example, if the environment variable **TERM** is set to **sun**, then the compiled terminal definition is found in **/usr/share/lib/terminfo/s/sun**. The **s** is copied from the first letter of **sun** to avoid creation of huge directories.) However, if **TERMINFO** is set to **$HOME/myterms**, **curses** will first check **$HOME/myterms/s/sun**, and, if that fails, will then check **/usr/share/lib/terminfo/s/sun**. This is useful for developing experimental definitions or when write permission on **/usr/share/lib/terminfo** is not available.

The integer variables **LINES** and **COLS** are defined in **<curses.h>**, and will be filled in by **initscr( )** with the size of the screen. For more information, see the subsection **Terminfo-Level Manipulations**. The constants **TRUE** and **FALSE** have the values **1** and **0**, respectively. The constants ERR and OK are returned by routines to indicate whether the routine successfully completed. These constants are also defined in **<curses.h>**.

ROUTINES

Many of the following routines have two or more versions. The routines prefixed with **w** require a *window* argument. The routines prefixed with **p** require a *pad* argument. Those without a prefix generally use **stdscr**.

The routines prefixed with **mv** require *y* and *x* coordinates to move to before performing the appropriate action. The **mv** routines imply a call to **move( )** before the call to the other routine. The window argument is always specified before the coordinates. *y* always refers to the row (of the window), and *x* always refers to the column. The upper left corner is always **(0,0)**, not **(1,1)**. The routines prefixed with **mvw** take both a *window* argument and *y* and *x* coordinates.

In each case, *win* is the window affected and *pad* is the pad affected. (**win** and *pad* are always of type **WINDOW *.**) Option-setting routines require a boolean flag *bf* with the value **TRUE** or **FALSE.** (*bf* is always of type **bool.**) The types **WINDOW**, **bool**, and **chtype** are defined in **<curses.h>** (see **SYNOPSIS** for a summary of what types all variables are).

All routines return either the integer ERR or the integer OK, unless otherwise noted. Routines that return pointers always return NULL on error.

Overall Screen Manipulation

WINDOW *initscr( )
The first routine called should almost always be **initscr( )**. The exceptions are **slk_init( )**, **filter( )**, and **ripoffline( )**. This will determine the terminal type and initialize all **curses** data structures. **initscr( )** also arranges that the first call to **refresh( )** will clear the screen. If errors occur, **initscr( )** will write an appropriate error message to standard error and exit; otherwise, a pointer to **stdscr** is returned. If the program wants an indication of error conditions, **newterm( )** should be used instead of **initscr( )**. **initscr( )** should only be called once per application.

endwin( )
A program should always call **endwin( )** before exiting or escaping from **curses** mode temporarily, to do a shell escape or **system**(3) call, for example. This routine will restore **termio**(4) modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call **wrefresh( )** or **doupdate( )**.

Window and Pad Manipulation

refresh( )

wrefresh (*win*)
These routines (or **prefresh( )**, **pnoutrefresh( )**, **wnoutrefresh( )**, or **doupdate( )**) must be called to write output to the terminal, as most other routines merely manipulate data structures. **wrefresh( )** copies the named window to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that's sent to the terminal (called optimization). **refresh( )** does the same thing, except it uses **stdscr** as a default window. Unless **leaveok( )** has been enabled, the physical cursor of the terminal is left at the location of the window's cursor. The number of characters output to the terminal is returned.

Note: **refresh( )** is a macro.

WINDOW *newwin (*nlines, ncols, begin_y, begin_x*)
Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The upper left corner of the window is at line *begin_y*, column *begin_x*. If either *nlines* or *ncols* is **0**, they will be set to the value of **lines**–*begin_y* and **cols**–*begin_x*. A new full-screen window is created by calling **newwin(0,0,0,0)**.

mvwin (*win, y, x*)
Move the window so that the upper left corner will be at position (*y*, *x*). If the move would cause the window to be off the screen, it is an error and the window is not moved.

WINDOW *subwin (*orig, nlines, ncols, begin_y, begin_x*)
Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The window is at position (*begin_y, begin_x*) on the screen. This position is relative to the screen, and not to the window *orig*. The window is made in the middle of the window *orig*, so that changes made to

one window will affect both windows. When using this routine, often it will be necessary to call **touchwin( )** or **touchline( )** on *orig* before calling **wrefresh**.

delwin (*win*) Delete the named window, freeing up all memory associated with it. In the case of overlapping windows, subwindows should be deleted before the main window.

Output

These routines are used to "draw" text on windows.

addch (*ch*)

waddch (*win, ch*)

mvaddch (*y, x, ch*)

mvwaddch (*win, y, x, ch*)

The character *ch* is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of **putchar( )** (see **putc**(3s)). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok( )** is enabled, the scrolling region will be scrolled up one line.

If *ch* is a TAB, NEWLINE, or backspace, the cursor will be moved appropriately within the window. A NEWLINE also does a **clrtoeol( )** before moving. TAB characters are considered to be at every eighth column. If *ch* is another control character, it will be drawn in the CTRL-X notation. (Calling **winch( )** after adding a control character will not return the control character, but instead will return the representation of the control character.)

Video attributes can be combined with a character by or-ing them into the parameter. This will result in these attributes also being set. The intent here is that text, including attributes, can be copied from one place to another using **inch( )** and **addch( )**. See **standout( )**, below.

Note: *ch* is actually of type **chtype**, not a character.

Note: **addch( )**, **mvaddch( )**, and **mvwaddch( )** are macros.

addstr (*str*)

waddstr (*win, str*)

mvwaddstr (*win, y, x, str*)

mvaddstr (*y, x, str*) These routines write all the characters of the null-terminated character string *str* on the given window. This is equivalent to calling **waddch( )** once for each character in the string.

Note: **addstr( )**, **mvaddstr( )**, and **mvwaddstr( )** are macros.

box (*win, vertch, horch*)

A box is drawn around the edge of the window, *win*. *vertch* and *horch* are the characters the box is to be drawn with. If *vertch* and *horch* are **0**, then appropriate default characters, **ACS_VLINE** and **ACS_HLINE**, will be used.

Note: *vertch* and *horch* are actually of type **chtype**, not characters.

erase( )

werase (*win*) These routines copy blanks to every position in the window.

Note: **erase( )** is a macro.

clear( )

wclear (*win*) These routines are like **erase**( ) and **werase**( ), but they also call **clearok**( ), arranging that the screen will be cleared completely on the next call to **wrefresh**( ) for that window, and repainted from scratch.

Note: **clear**( ) is a macro.

clrtobot( )

wclrtobot (*win*) All lines below the cursor in this window are erased. Also, the current line to the right of the cursor, inclusive, is erased.

Note: **clrtobot**( ) is a macro.

clrtoeol( )

wclrtoeol (*win*) The current line to the right of the cursor, inclusive, is erased.

Note: **clrtoeol**( ) is a macro.

delch( )

wdelch (*win*)

mvdelch (*y*, *x*)

mvwdelch (*win*, *y*, *x*) The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving to (*y*, *x*), if specified). This does not imply use of the hardware "delete-character" feature.

Note: **delch**( ), **mvdelch**( ), and **mvwdelch**( ) are macros.

deleteln( )

wdeleteln (*win*) The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. This does not imply use of the hardware "delete-line" feature.

Note: **deleteln**( ) is a macro.

getyx (*win*, *y*, *x*) The cursor position of the window is placed in the two integer variables *y* and *x*. This is implemented as a macro, so no '**&**' is necessary before the variables.

insch (*ch*)

winsch (*win*, *ch*)

mvwinsch (*win*, *y*, *x*, *ch*)

mvinsch (*y*, *x*, *ch*) The character *ch* is inserted before the character under the cursor. All characters to the right are moved one SPACE to the right, possibly losing the rightmost character of the line. The cursor position does not change (after moving to (*y*, *x*), if specified). This does not imply use of the hardware "insert-character" feature.

Note: *ch* is actually of type **chtype**, not a character.

Note: **insch**( ), **mvinsch**( ), and **mvwinsch**( ) are macros.

insertln( )

winsertln (*win*) A blank line is inserted above the current line and the bottom line is lost. This does not imply use of the hardware "insert-line" feature.

Note: **insertln**( ) is a macro.

move (*y*, *x*)

wmove (*win*, *y*, *x*) The cursor associated with the window is moved to line (row) *y*, column *x*. This does not move the physical cursor of the terminal until **refresh**( ) is called. The position specified is relative to the upper left corner of the window, which is (**0**, **0**).

Note: **move**( ) is a macro.

overlay (*srcwin*, *dstwin*)

overwrite (*srcwin*, *dstwin*)

These routines overlay *srcwin* on top of *dstwin*; that is, all text in *srcwin* is copied into *dstwin*. *scrwin* and *dstwin* need not be the same size; only text where the two windows overlap is copied. The difference is that **overlay**( ) is non-destructive (blanks are not copied), while **overwrite**( ) is destructive.

printw (*fmt* [, *arg*...])

wprintw (*win*, *fmt* [, arg...])

mvprintw (*y*, *x*, *fmt* [, *arg*...])

mvwprintw (*win*, *y*, *x*, *fmt* [, *arg*...])

These routines are analogous to **printf**(3V). The string that would be output by **printf**(3V) is instead output using **waddstr**( ) on the given window.

scroll (*win*) The window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the window is **stdscr** and the scrolling region is the entire window, the physical screen will be scrolled at the same time.

touchwin (*win*)

touchline (*win*, *start*, *count*)

Throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. **touchline**( ) only pretends that *count* lines have been changed, beginning with line *start*.

Input

getch( )

wgetch (*win*)

mvgetch (*y*, *x*)

mvwgetch (*win*, *y*, *x*) A character is read from the terminal associated with the window. In NODELAY mode, if there is no input waiting, the value ERR is returned. In DELAY mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak**( ), this will be after one character (CBREAK mode), or after the first newline (NOCBREAK mode). In HALF-DELAY mode, the program will hang until a character is typed or the specified timeout has been reached. Unless **noecho**( ) has been set, the character will also be echoed into the designated window. No **refresh**( ) will occur between the **move**( ) and the **getch**( ) done within the routines **mvgetch**( ) and **mvwgetch**( ).

When using **getch**( ), **wgetch**( ), **mvgetch**( ), or **mvwgetch**( ), do not set both NOCBREAK mode (**nocbreak**( )) and ECHO mode (**echo**( )) at the same time. Depending on the state of the terminal driver when each character is typed, the program may produce undesirable results.

If **keypad** (*win*, **TRUE**) has been called, and a function key is pressed, the token for that function key will be returned instead of the raw characters. See **keypad**() under **Input Options Setting**. Possible function keys are defined in **<curses.h>** with integers beginning with **0401**, whose names begin with **KEY_**. If a character is received that could be the beginning of a function key (such as escape), **curses** will set a timer. If the remainder of the sequence is not received within the designated time, the character will be passed through, otherwise the function key value will be returned. For this reason, on many terminals, there will be a delay after a user presses the escape key before the escape is returned to the program. Use by a programmer of the escape key for a single character routine is discouraged. Also see **notimeout**() below.

Note: **getch**(), **mvgetch**(), and **mvwgetch**() are macros.

getstr (*str*)

wgetstr (*win, str*)

mvgetstr (*y, x, str*)

mvwgetstr (*win, y, x, str*)

A series of calls to **getch**() is made, until a newline, carriage return, or enter key is received. The resulting value is placed in the area pointed at by the character pointer *str*. The user's erase and kill characters are interpreted. As in **mvgetch**(), no **refresh**() is done between the **move**() and **getstr**() within the routines **mvgetstr**() and **mvwgetstr**().

Note: **getstr**(), **mvgetstr**(), and **mvwgetstr**() are macros.

inch()

winch (*win*)

mvinch (*y, x*)

mvwinch (*win, y, x*)

The character, of type **chtype**, at the current position in the named window is returned. If any attributes are set for that position, their values will be OR'ed into the value returned. The predefined constants **A_CHARTEXT** and **A_ATTRIBUTES**, defined in **<curses.h>**, can be used with the C logical AND (&) operator to extract the character or attributes alone.

Note: **inch**(), **winch**(), **mvinch**(), and **mvwinch**() are macros.

scanw (*fmt* [,*arg* . . .])

wscanw (*win, fmt* [, *arg* . . .])

mvscanw (*y, x, fmt* [, *arg* . . .])

mvwscanw (*win, y, x, fmt* [, *arg* . . .])

These routines correspond to **scanf**(3V), as do their arguments and return values. **wgetstr**() is called on the window, and the resulting line is used as input for the scan.

Output Options Setting

These routines set options within **curses** that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling **endwin**().

clearok (*win, bf*)

If enabled (*bf* is **TRUE**), the next call to **wrefresh**() with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

idlok (*win, bf*) If enabled (*bf* is **TRUE**), **curses** will consider using the hardware "insert/delete-line" feature of terminals so equipped. If disabled (*bf* is **FALSE**), **curses** will very seldom use this feature. The "insert/delete-character" feature is always considered. This option should be enabled only if your application needs "insert/delete-line", for example, for a screen editor. It is disabled by default because "insert/delete-line" tends to be visually annoying when used in applications where it is not really needed. If "insert/delete-line" cannot be used, **curses** will redraw the changed portions of all lines.

leaveok (*win, bf*) Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

scrollok (*win, bf*) This option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (*bf* is **FALSE**), the cursor is left on the bottom line at the location where the offending character was entered. If enabled (*bf* is **TRUE**), **wrefresh( )** is called on the window, and then the physical terminal and window are scrolled up one line. Note: in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok( )**.

nl( )

nonl( ) These routines control whether NEWLINE is translated into RETURN and LINEFEED on output, and whether RETURN is translated into NEWLINE on input. Initially, the translations do occur. By disabling these translations using **nonl( )**, **curses** is able to make better use of the linefeed capability, resulting in faster cursor motion.

Input Options Setting

These routines set options within **curses** that deal with input. The options involve using **ioctl**(2) and therefore interact with **curses** routines. It is not necessary to turn these options off before calling **endwin( )**.

For more information on these options, refer to *Programming Utilities and Libraries.*

cbreak( )

nocbreak( ) These two routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. When in NOCBREAK mode, the tty driver will buffer characters typed until a NEWLINE or RETURN is typed. Interrupt and flow-control characters are unaffected by this mode (see **termio**(4)). Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should call **cbreak( )** or **nocbreak( )** explicitly. Most interactive programs using **curses** will set CBREAK mode.

Note: **cbreak( )** overrides **raw( )**. See **getch( )** under **Input** for a discussion of how these routines interact with **echo( )** and **noecho( )**.

echo( )

noecho( ) These routines control whether characters typed by the user are echoed by **getch( )** as they are typed. Echoing by the tty driver is always disabled, but initially **getch( )** is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho( )**. See **getch( )** under **Input** for a discussion of how these routines interact with **cbreak( )** and **nocbreak( )**.

raw( )

noraw( ) The terminal is placed into or out of RAW mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program. The differences are that in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal. RAW mode also causes 8-bit input and output. The behavior of the BREAK key depends on other bits in the terminal driver that are not set by **curses.**

Environment Queries

baudrate( ) Returns the output speed of the terminal. The number returned is in bits per second, for example, 9600, and is an integer.

char erasechar( ) The user's current erase character is returned.

char killchar( ) The user's current line-kill character is returned.

char *longname( ) This routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to **initscr( )** or **newterm( )**. The area is overwritten by each call to **newterm( )** and is not restored by **set_term( )**, so the value should be saved between calls to **newterm( ) if longname( )** is going to be used with multiple terminals.

Low-Level curses Access

The following routines give low-level access to various **curses** functionality. These routines typically would be used inside of library routines.

resetty( )

savetty( ) These routines save and restore the state of the terminal modes. **savetty( )** saves the current state of the terminal in a buffer and **resetty( )** restores the state to what it was at the last call to **savetty( )**.

Miscellaneous

unctrl (*c*) This macro expands to a character string which is a printable representation of the character *c*. Control characters are displayed in the ^X notation. Printing characters are displayed as is.

unctrl( ) is a macro, defined in **<unctrl.h>**, which is automatically included by **<curses.h>**.

flusok(*win,boolf*) set flush-on-refresh flag for *win*

getcap(*name*) get terminal capability *name*

touchoverlap(*win1,win2*)
mark overlap of *win1* on *win2* as changed

Use of curscr

The special window **curscr** can be used in only a few routines. If the window argument to **clearok( )** is **curscr**, the next call to **wrefresh( )** with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh( )** is **curscr**, the screen is immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" routine. The source window argument to **overlay( )**, **overwrite( )**, and **copywin** may be **curscr**, in which case the current contents of the virtual terminal screen will be accessed.

Obsolete Calls

Various routines are provided to maintain compatibility in programs written for older versions of the curses library. These routines are all emulated as indicated below.

crmode( ) Replaced by **cbreak( )**.

gettmode( ) A no-op.

nocrmode( ) Replaced by **nocbreak( )**.

SYSTEM V ROUTINES

The above routines are available as described except for **flusok( )**, **getcap( )** and **touchoverlap( )** which are not available.

In addition, the following routines are available:

Overall Screen Manipulation

isendwin( ) Returns **TRUE** if **endwin( )** has been called without any subsequent calls to **wrefresh( )**.

SCREEN *newterm(*type, outfd, infd*)

A program that outputs to more than one terminal must use **newterm( )** for each terminal instead of **initscr( )**. A program that wants an indication of error conditions, so that it may continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, must also use this routine. **newterm( )** should be called once for each terminal. It returns a variable of type **SCREEN*** that should be saved as a reference to that terminal. The arguments are the *type* of the terminal to be used in place of the environment variable **TERM**; *outfd*, a **stdio**(3V) file pointer for output to the terminal; and *infd*, another file pointer for input from the terminal. When it is done running, the program must also call **endwin( )** for each terminal being used. If **newterm( )** is called more than once for the same terminal, the first terminal referred to must be the last one for which **endwin( )** is called.

SCREEN *set_term (*new*)

This routine is used to switch between different terminals. The screen reference *new* becomes the new current terminal. A pointer to the screen of the previous terminal is returned by the routine. This is the only routine that manipulates **SCREEN** pointers; all other routines affect only the current terminal.

Window and Pad Manipulation

wnoutrefresh (*win*)

doupdate( ) These two routines allow multiple updates to the physical terminal screen with more efficiency than **wrefresh( )** alone. How this is accomplished is described in the next paragraph.

curses keeps two data structures representing the terminal screen: a *physical* terminal screen, describing what is actually on the screen, and a *virtual* terminal screen, describing what the programmer wants to have on the screen. **wrefresh( )** works by first calling **wnoutrefresh( )**, which copies the named window to the virtual screen, and then by calling **doupdate( )**, which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to output several windows at once, a series of calls to **wrefresh( )** will result in alternating calls to **wnoutrefresh( )** and **doupdate( )**, causing several bursts of output to the screen. By first calling **wnoutrefresh( )** for each window, it is then possible to call **doupdate( )** once, resulting in only one burst of output, with probably fewer total characters transmitted and certainly less processor time used.

WINDOW *newpad (*nlines, ncols*)

Create and return a pointer to a new pad data structure with the given number of lines (or rows), *nlines*, and columns, *ncols*. A pad is a window that is not restricted by the screen size and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (for example, from scrolling or echoing of input) do not occur. It is not legal to call **wrefresh( )** with a pad as an argument; the routines **prefresh( )** or **pnoutrefresh( )** should be called instead. Note: these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

WINDOW *subpad (*orig, nlines, ncols, begin_y, begin_x*)
Create and return a pointer to a subwindow within a pad with the given number of lines (or rows), *nlines*, and columns, *ncols*. Unlike **subwin( )**, which uses screen coordinates, the window is at position (*begin_y, begin_x*) on the pad. The window is made in the middle of the window *orig*, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call **touchwin( )** or **touchline( )** on *orig* before calling **prefresh( )**.

prefresh (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)

pnoutrefresh (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)

These routines are analogous to
wrefresh( ) and **wnoutrefresh( )** except that pads, instead of windows, are involved. The additional parameters are needed to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left corner, in the pad, of the rectangle to be displayed. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of *pminrow*, *pmincol*, *sminrow*, or *smincol* are treated as if they were zero.

Output

These routines are used to "draw" text on windows.

echochar (*ch*)

wechochar (*win, ch*)

pechochar (*pad, ch*)
These routines are functionally equivalent to a call to **addch** (*ch*) followed by a call to **refresh( )**, a call to **waddch** (*win, ch*) followed by a call to **wrefresh** (*win*), or a call to **waddch** (*pad, ch*) followed by a call to **prefresh** (*pad*). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain can be seen by using these routines instead of their equivalents. In the case of **pechochar( )**, the last location of the pad on the screen is reused for the arguments to **prefresh( )**.

Note: *ch* is actually of type **chtype**, not a character.

Note: **echochar( )** is a macro.

attroff (*attrs*)

wattroff (*win, attrs*)

attron (*attrs*)

wattron (*win, attrs*)

attrset (*attrs*)

wattrset (*win, attrs*)

beep( )

flash( ) These routines are used to signal the terminal user. **beep**( ) will sound the audible alarm on the terminal, if possible, and if not, will flash the screen (visible bell), if that is possible. **flash**( ) will flash the screen, and if that is not possible, will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep) but only some can flash the screen.

delay_output (*ms*) Insert a *ms* millisecond pause in the output. It is not recommended that this routine be used extensively, because padding characters are used rather than a processor pause.

getbegyx (*win, y, x*)

getmaxyx (*win, y, x*) Like **getyx**( ), these routines store the current beginning coordinates and size of the specified window.

Note: **getbegyx**( ) and **getmaxyx**( ) are macros.

copywin (*srcwin, dstwin, sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol, overlay*)

This routine provides a finer grain of control over the **overlay**( ) and **overwrite**( ) routines. Like in the **prefresh**( ) routine, a rectangle is specified in the destination window, (*dminrow, dmincol*) and (*dmaxrow, dmaxcol*), and the upper-left-corner coordinates of the source window, (*sminrow, smincol*). If the argument *overlay* is true, then copying is non-destructive, as in **overlay**( ).

vwprintw (*win, fmt, varglist*)

This routine corresponds to **vprintf**(3V). It performs a **wprintw**( ) using a variable argument list. The third argument is a **va_list**, a pointer to a list of arguments, as defined in **<varargs.h>**. See the **vprintf**(3V) and **varargs**(3) manual pages for a detailed description on how to use variable argument lists.

Input

flushinp( ) Throws away any typeahead that has been typed by the user and has not yet been read by the program.

ungetch (*c*) Place *c* back onto the input queue to be returned by the next call to **wgetch**( ).

vwscanw (*win, fmt, ap*) This routine is similar to **vwprintw**( ) above in that performs a **wscanw**( ) using a variable argument list. The third argument is a **va_list**, a pointer to a list of arguments, as defined in **<varargs.h>**. See the **vprintf**(3V) and **varargs**(3) manual pages for a detailed description on how to use variable argument lists.

Output Options Setting

These routines set options within **curses** that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling **endwin**( ).

setscrreg (*top, bot*)

wsetscrreg (*win, top, bot*)

These routines allow the user to set a software scrolling region in a window. *top* and *bot* are the line numbers of the top and bottom margin of the scrolling region. Line 0 is the top line of the window. If this option and **scrollok**( ) are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. Note: this has nothing to do with use of a physical scrolling region capability in the terminal, like that in the DEC VT100. Only the text of the window is scrolled; if **idlok()** is enabled and the terminal has either a scrolling region or "insert/delete-line" capability, they will probably be used by the output routines.

Note: **setscrreg()** and **wsetscrreg()** are macros.

Input Options Setting

These routines set options within **curses** that deal with input. The options involve using **ioctl**(2) and therefore interact with **curses** routines. It is not necessary to turn these options off before calling **endwin()**.

For more information on these options, refer to *Programming Utilities and Libraries.*

halfdelay (*tenths*)
: Half-delay mode is similar to CBREAK mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, ERR will be returned if nothing has been typed. *tenths* must be a number between 1 and 255. Use **nocbreak()** to leave half-delay mode.

intrflush (*win, bf*)
: If this option is enabled, when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing **curses** to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

keypad (*win, bf*)
: This option enables the keypad of the user's terminal. If enabled, the user can press a function key (such as an arrow key) and **wgetch()** will return a single value representing the function key, as in **KEY_LEFT**. If disabled, **curses** will not treat function keys specially and the program would have to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option will cause the terminal keypad to be turned on when **wgetch()** is called.

meta (*win, bf*)
: If enabled, characters returned by **wgetch()** are transmitted with all 8 bits, instead of with the highest bit stripped. In order for **meta()** to work correctly, the **km** (**has_meta_key**) capability has to be specified in the terminal's **terminfo**(5V) entry.

nodelay (*win, bf*)
: This option causes **wgetch()** to be a non-blocking call. If no input is ready, **wgetch()** will return ERR. If disabled, **wgetch()** will hang until a key is pressed.

notimeout (*win, bf*)
: While interpreting an input escape sequence, **wgetch()** will set a timer while waiting for the next character. If **notimeout** (*win,* **TRUE**) is called, then **wgetch()** will not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

typeahead (*fildes*)
: **curses** does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update will be postponed until **refresh()** or **doupdate()** is called again. This allows faster response to commands typed in advance. Normally, the file descriptor for the input FILE pointer passed to **newterm()**, or **stdin** in the case that **initscr()** was used, will be used to do this typeahead checking. The **typeahead()** routine specifies that the file descriptor *fildes* is to be used to check for typeahead instead. If *fildes* is **–1**, then no typeahead checking will be done.

Note: *fildes* is a file descriptor, not a **<stdio.h>** FILE pointer.

Environment Queries

has_ic( ) True if the terminal has insert- and delete-character capabilities.

has_il() True if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to check to see if it would be appropriate to turn on physical scrolling using **scrollok( )**.

Soft Labels

If desired, **curses** will manipulate the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, if you want to simulate them, **curses** will take over the bottom line of **stdscr**, reducing the size of **stdscr** and the variable **LINES**. **curses** standardizes on 8 labels of 8 characters each.

slk_init (*labfmt*) In order to use soft labels, this routine must be called before **initscr( )** or **newterm( )** is called. If **initscr( )** winds up using a line from **stdscr** to emulate the soft labels, then *labfmt* determines how the labels are arranged on the screen. Setting *labfmt* to **0** indicates that the labels are to be arranged in a 3-2-3 arrangement; **1** asks for a 4-4 arrangement.

slk_set (*labnum, label, labfmt*)
labnum is the label number, from 1 to 8. *label* is the string to be put on the label, up to 8 characters in length. A null string or a NULL pointer will put up a blank label. *labfmt* is one of **0**, **1** or **2**, to indicate whether the label is to be left-justified, centered, or right-justified within the label.

slk_refresh( )

slk_noutrefresh( ) These routines correspond to the routines **wrefresh( )** and **wnoutrefresh( )**. Most applications would use **slk_noutrefresh( )** because a **wrefresh( )** will most likely soon follow.

char *slk_label (*labnum*)
The current label for label number *labnum*, with leading and trailing blanks stripped, is returned.

slk_clear( ) The soft labels are cleared from the screen.

slk_restore( ) The soft labels are restored to the screen after a **slk_clear( )**.

slk_touch( ) All of the soft labels are forced to be output the next time a **slk_noutrefresh( )** is performed.

Low-Level curses Access

The following routines give low-level access to various **curses** functionality. These routines typically would be used inside of library routines.

def_prog_mode( )

def_shell_mode( ) Save the current terminal modes as the "program" (in **curses**) or "shell" (not in **curses**) state for use by the **reset_prog_mode( )** and **reset_shell_mode( )** routines. This is done automatically by **initscr( )**.

reset_prog_mode( )

reset_shell_mode( ) Restore the terminal to "program" (in **curses**) or "shell" (out of **curses**) state. These are done automatically by **endwin( )** and **doupdate( )** after an **endwin( )**, so they normally would not be called.

getsyx (*y*, *x*)
The current coordinates of the virtual screen cursor are returned in *y* and *x*. Like **getyx()**, the variables *y* and *x* do not take an **&** before them. If **leaveok()** is currently **TRUE**, then **–1**, **–1** will be returned. If lines may have been removed from the top of the screen using **ripoffline()** and the values are to be used beyond just passing them on to **setsyx()**, the value *y*+**stdscr–>_yoffset** should be used for those other uses.

Note: **getsyx()** is a macro.

setsyx (*y*, *x*)
The virtual screen cursor is set to *y*, *x*. If *y* and *x* are both –1, then **leaveok()** will be set. The two routines **getsyx()** and **setsyx()** are designed to be used by a library routine that manipulates **curses** windows but does not want to mess up the current position of the program's cursor. The library routine would call **getsyx()** at the beginning, do its manipulation of its own windows, do a **wnoutrefresh()** on its windows, call **setsyx()**, and then call **doupdate()**.

ripoffline (*line*, *init*)
This routine provides access to the same facility that **slk_init()** uses to reduce the size of the screen. **ripoffline()** must be called before **initscr()** or **newterm()** is called. If *line* is positive, a line will be removed from the top of **stdscr**; if negative, a line will be removed from the bottom. When this is done inside **initscr()**, the routine *init* is called with two arguments: a window pointer to the 1-line window that has been allocated and an integer with the number of columns in the window. Inside this initialization routine, the integer variables **LINES** and **COLS** (defined in **<curses.h>**) are not guaranteed to be accurate and **wrefresh()** or **doupdate()** must not be called. It is allowable to call **wnoutrefresh()** during the initialization routine.

ripoffline() can be called up to five times before calling **initscr()** or **newterm()**.

scr_dump (*filename*)
The current contents of the virtual screen are written to the file *filename*.

scr_restore (*filename*)
The virtual screen is set to the contents of *filename*, which must have been written using **scr_dump()**. The next call to **doupdate()** will restore the screen to what it looked like in the dump file.

scr_init (*filename*)
The contents of *filename* are read in and used to initialize the **curses** data structures about what the terminal currently has on its screen. If the data is determined to be valid, **curses** will base its next update of the screen on this information rather than clearing the screen and starting from scratch. **scr_init()** would be used after **initscr()** or a **system**(3) call to share the screen with another process that has done a **scr_dump()** after its **endwin()** call. The data will be declared invalid if the time-stamp of the tty is old or the **terminfo**(5V) capability **nrrmc** is true.

curs_set (*visibility*)
The cursor is set to invisible, normal, or very visible for *visibility* equal to **0**, **1** or **2**.

draino (*ms*)
Wait until the output has drained enough that it will only take *ms* more milliseconds to drain completely.

garbagedlines (*win*, *begline*, *numlines*)
This routine indicates to **curses** that a screen line is garbaged and should be thrown away before having anything written over the top of it. It could be used for programs such as editors that want a command to redraw just a single line. Such a command could be used in cases where there is a noisy communications line and redrawing the entire screen would be subject to even more communication noise. Just redrawing the single line gives some semblance of hope that it would show up unblemished. The current location of the window is used to determine which lines are to be redrawn.

napms (*ms*) Sleep for *ms* milliseconds.

Terminfo-Level Manipulations

These low-level routines must be called by programs that need to deal directly with the **terminfo**(5V) database to handle certain terminal capabilities, such as programming function keys. For all other functionality, **curses** routines are more suitable and their use is recommended.

Initially, **setupterm**( ) should be called. Note: **setupterm**( ) is automatically called by **initscr**( ) and **newterm**( ). This will define the set of terminal-dependent variables defined in the **terminfo**(5V) database. The **terminfo**(5V) variables *lines* and *columns* (see **terminfo**(5V)) are initialized by **setupterm**( ) as follows: if the environment variables **LINES** and **COLUMNS** exist, their values are used. If the above environment variables do not exist, and the window sizes in rows and columns as returned by the **TIOCGWINSZ ioctl** are non-zero, those sizes are used. Otherwise, the values for *lines* and *columns* specified in the **terminfo**(5V) database are used.

The header files **<curses.h>** and **<term.h>** should be included, in this order, to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through **tparm**( ) to instantiate them. All **terminfo**(5V) strings (including the output of **tparm**( ) should be printed with **tputs**( ) or **putp**( ). Before exiting, **reset_shell_mode**( ) should be called to restore the tty modes. Programs that use cursor addressing should output **enter_ca_mode** upon startup and should output **exit_ca_mode** before exiting (see **terminfo**(5V)). Programs desiring shell escapes should call **reset_shell_mode**( ) and output **exit_ca_mode** before the shell is called and should output **enter_ca_mode** and call **reset_prog_mode**( ) after returning from the shell. Note: this is different from the **curses** routines (see **endwin**( )).

setupterm (*term, fildes, errret*)

Reads in the **terminfo**(5V) database, initializing the **terminfo**(5V) structures, but does not set up the output virtualization structures used by **curses**. The terminal type is in the character string *term*; if *term* is NULL, the environment variable **TERM** will be used. All output is to the file descriptor *fildes*. If *errret* is not NULL, then **setupterm**( ) will return OK or **ERR** and store a status value in the integer pointed to by *errret*. A status of **1** in *errret* is normal, **0** means that the terminal could not be found, and **–1** means that the **terminfo**(5V) database could not be found. If *errret* is NULL, **setupterm**( ) will print an error message upon finding an error and exit. Thus, the simplest call is '**setupterm ((char *)0, 1, (int *)0)**', which uses all the defaults.

The **terminfo**(5V) boolean, numeric and string variables are stored in a structure of type **TERMINAL**. After **setupterm**( ) returns successfully, the variable *cur_term* (of type **TERMINAL ***) is initialized with all of the information that the **terminfo**(5V) boolean, numeric and string variables refer to. The pointer may be saved before calling **setupterm**( ) again. Further calls to **setupterm**( ) will allocate new space rather than reuse the space pointed to by *cur_term*.

set_curterm (*nterm*) *nterm* is of type **TERMINAL ***. **set_curterm**( ) sets the variable *cur_term* to *nterm*, and makes all of the **terminfo**(5V) boolean, numeric and string variables use the values from *nterm*.

del_curterm (*oterm*) *oterm* is of type **TERMINAL ***. **del_curterm**( ) frees the space pointed to by *oterm* and makes it available for further use. If *oterm* is the same as *cur_term*, then references to any of the **terminfo**(5V) boolean, numeric and string variables thereafter may refer to invalid memory locations until another **setupterm**( ) has been called.

restartterm (*term, fildes, errret*)

Like **setupterm**( ) after a memory restore.

char *tparm ($str, p_1, p_2, \ldots, p_9$)

Instantiate the string *str* with parms p_i. A pointer is returned to the result of *str* with the parameters applied.

tputs (*str, count, putc*) Apply padding to the string *str* and output it. *str* must be a **terminfo**(5V) string variable or the return value from **tparm**( ), **tgetstr**( ), **tigetstr**( ) or **tgoto**( ). *count* is the number of lines affected, or **1** if not applicable. **putchar**( ) is a **putc**(3s)-like routine to which the characters are passed, one at a time.

putp (*str*) A routine that calls **tputs**( ) (*str*, **1**, **putc(3s).**

vidputs (*attrs, putc*) Output a string that puts the terminal in the video attribute mode *attrs*, which is any combination of the attributes listed below. The characters are passed to the **putc**(3s)-like routine **putc**(3s).

vidattr (*attrs*) Like **vidputs**( ), except that it outputs through **putc**(3s).

tigetflag (*capname*) The value **−1** is returned if *capname* is not a boolean capability.

tigetnum (*capname*) The value **−2** is returned if *capname* is not a numeric capability.

tigetstr (*capname*) The value **(char *) −1** is returned if *capname* is not a string capability.

Termcap Emulation

These routines are included as a conversion aid for programs that use the **termcap**(3X) library. Their parameters are the same and the routines are emulated using the **terminfo**(5V) database.

tgetent (*bp, name*) Look up **termcap** entry for *name*. The emulation ignores the buffer pointer *bp*.

tgetflag (*codename*) Get the boolean entry for *codename*.

tgetnum (*codes*) Get numeric entry for *codename*.

char *tgetstr (*codename, area*)
Return the string entry for *codename*. If *area* is not NULL, then also store it in the buffer pointed to by *area* and advance *area*. **tputs**( ) should be used to output the returned string.

char *tgoto (*cap, col, row*)
Instantiate the parameters into the given capability. The output from this routine is to be passed to **tputs**( ).

tputs (*str, affcnt, putc*) See **tputs**( ) above, under **Terminfo-Level Manipulations.**

Miscellaneous

char *keyname (*c*) A character string corresponding to the key *c* is returned.

filter( ) This routine is one of the few that is to be called before **initscr**( ) or **newterm**( ) is called. It arranges things so that **curses** thinks that there is a 1-line screen. **curses** will not use any terminal capabilities that assume that they know what line on the screen the cursor is on.

Use of curscr

The special window **curscr** can be used in only a few routines. If the window argument to **clearok**( ) is **curscr**, the next call to **wrefresh**( ) with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh**( ) is **curscr**, the screen is immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" routine. The source window argument to **overlay**( ), **overwrite**( ), and **copywin** may be **curscr**, in which case the current contents of the virtual terminal screen will be accessed.

Obsolete Calls

Various routines are provided to maintain compatibility in programs written for older versions of the curses library. These routines are all emulated as indicated below.

crmode( ) Replaced by **cbreak**( ).

fixterm( ) Replaced by **reset_prog_mode**( ).

nocrmode( ) Replaced by **nocbreak**( ).

resetterm( )	Replaced by **reset_shell_mode( )**.
saveterm( )	Replaced by **def_prog_mode( )**.
setterm( )	Replaced by **setupterm( )**.

SYSTEM V ATTRIBUTES

The following video attributes, defined in <**curses.h**>, can be passed to the routines **attron( )**, **attroff( )**, and **attrset( )**, or OR'ed with the characters passed to **addch( )**.

A_STANDOUT	Terminal's best highlighting mode
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_ALTCHARSET	Alternate character set
A_CHARTEXT	Bit-mask to extract character (described under **winch**)
A_ATTRIBUTES	Bit-mask to extract attributes (described under **winch**)
A_NORMAL	Bit mask to reset all attributes off (for example: '**attrset (A_NORMAL)**'

SYSTEM V FUNCTION KEYS

The following function keys, defined in <**curses.h**>, might be returned by **getch( )** if **keypad( )** has been enabled. Note: not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or the definition for the key is not present in the **terminfo**(5V) database.

Name	*Value*	*Key name*
KEY_BREAK	0401	break key (unreliable)
KEY_DOWN	0402	The four arrow keys ...
KEY_UP	0403	
KEY_LEFT	0404	
KEY_RIGHT	0405	...
KEY_HOME	0406	Home key (upward+left arrow)
KEY_BACKSPACE	0407	backspace (unreliable)
KEY_F0	0410	Function keys. Space for 64 keys is reserved.
KEY_F(n)	(KEY_F0+(n))	Formula for f_n.
KEY_DL	0510	Delete line
KEY_IL	0511	Insert line
KEY_DC	0512	Delete character
KEY_IC	0513	Insert char or enter insert mode
KEY_EIC	0514	Exit insert char mode
KEY_CLEAR	0515	Clear screen
KEY_EOS	0516	Clear to end of screen
KEY_EOL	0517	Clear to end of line
KEY_SF	0520	Scroll 1 line forward
KEY_SR	0521	Scroll 1 line backwards (reverse)
KEY_NPAGE	0522	Next page
KEY_PPAGE	0523	Previous page
KEY_STAB	0524	Set TAB
KEY_CTAB	0525	Clear TAB
KEY_CATAB	0526	Clear all TAB characters
KEY_ENTER	0527	Enter or send
KEY_SRESET	0530	soft (partial) reset

KEY_RESET	0531	reset or hard reset
KEY_PRINT	0532	print or copy
KEY_LL	0533	home down or bottom (lower left) keypad is arranged like this: A1 up A3 left B2 right C1 down C3
KEY_A1	0534	Upper left of keypad
KEY_A3	0535	Upper right of keypad
KEY_B2	0536	Center of keypad
KEY_C1	0537	Lower left of keypad
KEY_C3	0540	Lower right of keypad
KEY_BTAB	0541	Back TAB key
KEY_BEG	0542	beg(inning) key
KEY_CANCEL	0543	cancel key
KEY_CLOSE	0544	close key
KEY_COMMAND	0545	cmd (command) key
KEY_COPY	0546	copy key
KEY_CREATE	0547	create key
KEY_END	0550	end key
KEY_EXIT	0551	exit key
KEY_FIND	0552	find key
KEY_HELP	0553	help key
KEY_MARK	0554	mark key
KEY_MESSAGE	0555	message key
KEY_MOVE	0556	move key
KEY_NEXT	0557	next object key
KEY_OPEN	0560	open key
KEY_OPTIONS	0561	options key
KEY_PREVIOUS	0562	previous object key
KEY_REDO	0563	redo key
KEY_REFERENCE	0564	ref(erence) key
KEY_REFRESH	0565	refresh key
KEY_REPLACE	0566	replace key
KEY_RESTART	0567	restart key
KEY_RESUME	0570	resume key
KEY_SAVE	0571	save key
KEY_SBEG	0572	shifted beginning key
KEY_SCANCEL	0573	shifted cancel key
KEY_SCOMMAND	0574	shifted command key
KEY_SCOPY	0575	shifted copy key
KEY_SCREATE	0576	shifted create key
KEY_SDC	0577	shifted delete char key
KEY_SDL	0600	shifted delete line key
KEY_SELECT	0601	select key
KEY_SEND	0602	shifted end key
KEY_SEOL	0603	shifted clear line key
KEY_SEXIT	0604	shifted exit key
KEY_SFIND	0605	shifted find key
KEY_SHELP	0606	shifted help key
KEY_SHOME	0607	shifted home key
KEY_SIC	0610	shifted input key
KEY_SLEFT	0611	shifted left arrow key

KEY_SMESSAGE	0612	shifted message key
KEY_SMOVE	0613	shifted move key
KEY_SNEXT	0614	shifted next key
KEY_SOPTIONS	0615	shifted options key
KEY_SPREVIOUS	0616	shifted prev key
KEY_SPRINT	0617	shifted print key
KEY_SREDO	0620	shifted redo key
KEY_SREPLACE	0621	shifted replace key
KEY_SRIGHT	0622	shifted right arrow
KEY_SRSUME	0623	shifted resume key
KEY_SSAVE	0624	shifted save key
KEY_SSUSPEND	0625	shifted suspend key
KEY_SUNDO	0626	shifted undo key
KEY_SUSPEND	0627	suspend key
KEY_UNDO	0630	undo key

LINE GRAPHICS

The following variables may be used to add line-drawing characters to the screen with **waddce**. When defined for the terminal, the variable will have the **A_ALTCHARSET** bit turned on. Otherwise, the default character listed below will be stored in the variable. The names were chosen to be consistent with the DEC VT100 nomenclature.

Name	*Default*	*Glyph Description*
ACS_ULCORNER	+	upper left corner
ACS_LLCORNER	+	lower left corner
ACS_URCORNER	+	upper right corner
ACS_LRCORNER	+	lower right corner
ACS_RTEE	+	right tee (┤)
ACS_LTEE	+	left tee (├)
ACS_BTEE	+	bottom tee (┴)
ACS_TTEE	+	top tee (┬)
ACS_HLINE	–	horizontal line
ACS_VLINE	|	vertical line
ACS_PLUS	+	plus
ACS_S1	–	scan line 1
ACS_S9	_	scan line 9
ACS_DIAMOND	+	diamond
ACS_CKBOARD	:	checker board (stipple)
ACS_DEGREE	'	degree symbol
ACS_PLMINUS	#	plus/minus
ACS_BULLET	o	bullet
ACS_LARROW	<	arrow pointing left
ACS_RARROW	>	arrow pointing right
ACS_DARROW	v	arrow pointing down
ACS_UARROW	^	arrow pointing up
ACS_BOARD	#	board of squares
ACS_LANTERN	#	lantern symbol
ACS_BLOCK	#	solid square block

RETURN VALUES

Unless otherwise noted in the preceding routine descriptions, all routines return:

OK on success.

ERR on failure.

SYSTEM V RETURN VALUES

All macros return the value of their **w** version, except **setscrreg( )**, **wsetscrreg( )**, **getsyx( )**, **getyx( )**, **getbegy( )**, **getmaxyx( )**, which return no useful value.

Routines that return pointers always return (*type* *) NULL on failure.

FILES

.login

.profile

SYSTEM V FILES

/usr/share/lib/terminfo

SEE ALSO

cc(1V), **ld**(1), **ioctl**(2), **getenv**(3V), **plot**(3X), **printf**(3V), **putc**(3S), **scanf**(3V), **stdio**(3V), **system**(3), **varargs**(3), **vprintf**(3V), **termio**(4), **tty**(4), **term**(5V), **termcap**(5), **terminfo**(5V), **tic**(8V)

SYSTEM V WARNINGS

The plotting library **plot**(3X) and the curses library **curses**(3V) both use the names **erase( )** and **move( )**. The **curses** versions are macros. If you need both libraries, put the **plot**(3X) code in a different source file than the **curses**(3V) code, and/or '**#undef move**' and '**#undef erase**' in the **plot**(3X) code.

Between the time a call to **initscr( )** and **endwin( )** has been issued, use only the routines in the **curses** library to generate output. Using system calls or the "standard I/O package" (see **stdio**(3V)) for output during that time can cause unpredictable results.

NAME

cuserid – get character login name of the user

SYNOPSIS

```
#include <stdio.h>

char *cuserid(s)
char *s;
```

DESCRIPTION

cuserid() returns a pointer to a string representing the login name under which the owner of the current process is logged in. If *s* is a NULL pointer, this string is placed in an internal static area, the address of which is returned. Otherwise, *s* is assumed to point to an array of at least **L_cuserid** characters; the representation is left in this array. The constant **L_cuserid** is defined in the **<stdio.h>** header file.

SEE ALSO

cc(1V), **ld**(1), **getlogin**(3V), **getpwent**(3V)

RETURN VALUES

cuserid() returns a pointer to the login name on success. On failure, **cuserid()** returns NULL, and if *s* is not NULL, places a null character (**'\0'**) at **s[0]**.

NOTES

The internal static area to which **cuserid()** writes when *s* is NULL will be overwritten by a subsequent call to **getpwnam()** (see **getpwent**(3V)).

A compatibility problem has been identified with the **cuserid()** function. The traditional version of this library routine in SunOS Release 3.2 and later releases and all System V releases calls the **getlogin()** function, and if it fails uses the **getpwuid()** function to try to return a name associated with the real user ID associated with the calling process. POSIX.1 requires that the **cuserid()** function try to return a name associated with the effective user ID associated with the calling process. Although this usually yields the same results, use of set-uid programs may yield different results.

A binding interpretation has been issued by IEEE saying that the POSIX.1 functionality has to be provided for compliance with POSIX.1. However, balloting on the first update to POSIX.1, P1003.1a, has led to the removal of the **cuserid()** function from the standard. (This is the state in the second recirculation ballot of P1003.1a dated 11 December 1989.) The objections leading to this resolution had both users and implementors arguing for the historical version and for the version specified by POSIX.1. The only way to reach consensus appears to be to remove the function from the standard.

To further complicate the issue, System V Release 4.0 has kept the traditional version of **cuserid()**. XPG3 specifies the POSIX.1 version of **cuserid()**, but the test suite for conformance to XPG3 promises to accept either implementation. Both of these are anticipating the final approval of P1003.1a as a standard with the **cuserid()** function removed. Since we also expect the **cuserid()** function to be dropped from the standard when P1003.1a is approved, SunOS Release 4.1 provides the traditional **cuserid()** function in the C library. However, for users that need the version specified by POSIX.1, it is provided in a POSIX library available in the System V environment. This library can be accessed by specifying **–lposix** on the **cc**(1V) or **ld**(1) command line.

NAME

dbm, dbminit, dbmclose, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNOPSIS

```
#include <dbm.h>

typedef struct {
        char *dptr;
        int dsize;
} datum;

dbminit(file)
char *file;

dbmclose( )

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey( )

datum nextkey(key)
datum key;
```

DESCRIPTION

Note: the **dbm( )** library has been superceded by **ndbm**(3), and is now implemented using **ndbm( )**.

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **–ldbm**.

*key*s and *content*s are described by the **datum** typedef. A **datum** specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by **dbminit**. At the time of this call, the files *file*.**dir** and *file*.**pag** must exist. (An empty database is created by creating zero-length **.dir** and **.pag** files.)

A database may be closed by calling **dbmclose**. You must close a database before opening a new one.

Once open, the data stored under a key is accessed by **fetch( )** and data is placed under a key by **store**. A key (and its associated contents) is deleted by **delete**. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **firstkey( )** and **nextkey**. **firstkey( )** will return the first key in the database. With any key **nextkey( )** will return the next key in the database. This code will traverse the data base:

```
for (key = firstkey( ); key.dptr != NULL; key = nextkey(key))
```

SEE ALSO

ar(1V), **cat**(1V), **cp**(1), **tar**(1), **ndbm**(3)

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero return indicates no error. Routines that return a **datum** indicate errors with a NULL **(0)** *dptr*.

BUGS

The **.pag** file will contain holes so that its apparent size is about four times its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (**cp**(1), **cat**(1V), **tar**(1), **ar**(1V)) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. **store( )** will return an error in the event that a disk block fills with inseparable data.

delete( ) does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **firstkey( )** and **nextkey( )** depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

NAME

decimal_to_single, decimal_to_double, decimal_to_extended – convert decimal record to floating-point value

SYNOPSIS

#include <floatingpoint.h>

void decimal_to_single(px, pm, pd, ps)
single *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void decimal_to_double(px, pm, pd, ps)
double *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void decimal_to_extended(px, pm, pd, ps)
extended *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

DESCRIPTION

The **decimal_to_floating()** functions convert the decimal record at **pd* into a floating-point value at **px*, observing the modes specified in **pm* and setting exceptions in **ps*. If there are no IEEE exceptions, **ps* will be zero.

pd->sign and *pd->fpclass* are always taken into account. *pd->exponent* and *pd->ds* are used when *pd->fpclass* is *fp_normal* or *fp_subnormal*. In these cases *pd->ds* must contain one or more ascii digits followed by a null character. **px* is set to a correctly rounded approximation to

(pd->sign)*(pd->ds)*10(pd->exponent)**

Thus if *pd->exponent* == –2 and *pd->ds* == "1234", **px* will get 12.34 rounded to storage precision. *pd->ds* cannot have more than **DECIMAL_STRING_LENGTH-1** significant digits because one character is used to terminate the string with a null character. If *pd->more != 0* on input then additional nonzero digits follow those in *pd->ds; fp_inexact* is set accordingly on output in **ps*.

**px* is correctly rounded according to the IEEE rounding modes in *pm->rd*. **ps* is set to contain *fp_inexact*, *fp_underflow*, or *fp_overflow* if any of these arise.

pd->ndigits, *pm->df*, and *pm->ndigits* are not used.

strtod(3), **scanf**(3V), **fscanf**( ), and **sscanf**( ) all use **decimal_to_double**( ).

SEE ALSO

scanf(3V), **strtod**(3)

NAME

des_crypt, ecb_crypt, cbc_crypt, des_setparity – fast DES encryption

SYNOPSIS

```
#include <des_crypt.h>

int ecb_crypt(key, data, datalen, mode)
char *key;
char *data;
unsigned datalen;
unsigned mode;

int cbc_crypt(key, data, datalen, mode, ivec)
char *key;
char *data;
unsigned datalen;
unsigned mode;
char *ivec;

void des_setparity(key)
char *key;
```

DESCRIPTION

ecb_crypt() and **cbc_crypt()** implement the NBS DES (Data Encryption Standard). These routines are faster and more general purpose than **crypt**(3). They also are able to utilize DES hardware if it is available. **ecb_crypt()** encrypts in ECB (Electronic Code Book) mode, which encrypts blocks of data independently. **cbc_crypt()** encrypts in CBC (Cipher Block Chaining) mode, which chains together successive blocks. CBC mode protects against insertions, deletions and substitutions of blocks. Also, regularities in the clear text will not appear in the cipher text.

Here is how to use these routines. The first parameter, *key*, is the 8-byte encryption key with parity. To set the key's parity, which for DES is in the low bit of each byte, use *des_setparity*. The second parameter, *data*, contains the data to be encrypted or decrypted. The third parameter, *datalen*, is the length in bytes of *data*, which must be a multiple of 8. The fourth parameter, *mode*, is formed by OR'ing together some things. For the encryption direction 'or' in either DES_ENCRYPT or DES_DECRYPT. For software versus hardware encryption, 'or' in either DES_HW or DES_SW. If DES_HW is specified, and there is no hardware, then the encryption is performed in software and the routine returns DESERR_NOHWDEVICE. For *cbc_crypt*, the parameter *ivec* is the 8-byte initialization vector for the chaining. It is updated to the next initialization vector upon return.

SEE ALSO

des(1), **crypt**(3)

DIAGNOSTICS

DESERR_NONE	No error.
DESERR_NOHWDEVICE	
	Encryption succeeded, but done in software instead of the requested hardware.
DESERR_HWERR	An error occurred in the hardware or driver.
DESERR_BADPARAM	Bad parameter to routine.

Given a result status *stat*, the macro DES_FAILED(stat) is false only for the first two statuses.

RESTRICTIONS

These routines are not available for export outside the U.S.

NAME

directory, opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

#include <dirent.h>

DIR *opendir(dirname)
char *dirname;

struct dirent *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

void seekdir(dirp, loc)
DIR *dirp;
long loc;

void rewinddir(dirp)
DIR *dirp;

int closedir(dirp)
DIR *dirp;

SYSTEM V SYNOPSIS

For XPG2 conformance, use:

#include <sys/dirent.h>

DESCRIPTION

opendir() opens the directory named by *dirname* and associates a *directory stream* with it. **opendir()** returns a pointer to be used to identify the directory stream in subsequent operations. A NULL pointer is returned if *dirname* cannot be accessed or is not a directory, or if it cannot **malloc**(3V) enough memory to hold the whole thing.

readdir() returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid **seekdir()** operation.

telldir() returns the current location associated with the named directory stream.

seekdir() sets the position of the next **readdir()** operation on the directory stream. The new position reverts to the one associated with the directory stream when the **telldir()** operation was performed. Values returned by **telldir()** are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the **telldir()** value may be invalidated due to undetected directory compaction. It is safe to use a previous **telldir()** value immediately after a call to **opendir()** and before any calls to **readdir**.

rewinddir() resets the position of the named directory stream to the beginning of the directory. I also causes the directory stream to refer to the current state of the corresponding directory, as a call to **opendir()** would have done.

closedir() closes the named directory stream and frees the structure associated with the DIR pointer.

RETURN VALUES

opendir( ) returns a pointer to an object of type **DIR** on success. On failure, it returns NULL and sets **errno** to indicate the error.

readdir( ) returns a pointer to an object of type **struct dirent** on success. On failure, it returns NULL and sets **errno** to indicate the error. When the end of the directory is encountered, **readdir**( ) returns NULL and leaves **errno** unchanged.

closedir( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

telldir( ) returns the current location associated with the specified directory stream.

ERRORS

If any of the following conditions occur, **opendir**( ) sets **errno** to:

EACCES	Search permission is denied for a component of *dirname*.
	Read permission is denied for *dirname*.
ENAMETOOLONG	The length of *dirname* exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The named directory does not exist.
ENOTDIR	A component of *dirname* is not a directory.

for each of the following conditions, when the condition is detected, **opendir**( ) sets **errno** to one of the following:

EMFILE	Too many file descriptors are currently open for the process.
ENFILE	Too many file descriptors are currently open in the system.

For each of the following conditions, when the condition is detected, **readdir**( ) sets **errno** to the following:

EBADF	*dirp* does not refer to an open directory stream.

For each of the following conditions, when the condition is detected, **closedir**( ) sets **errno** to the following:

EBADF	*dirp* does not refer to an open directory stream.

SYSTEM V ERRORS

In addition to the above, **opendir**( ) may set **errno** to the following:

ENOENT	*dirname* points to an empty string.

EXAMPLES

Sample code which searchs a directory for entry ''name'' is:

```
dirp = opendir(".");
for (dp = readdir(dirp); dp != NULL; dp = readdir(dirp))
        if (!strcmp(dp->d_name, name)) {
                closedir (dirp);
                return FOUND;
        }
closedir (dirp);
return NOT_FOUND;
```

SEE ALSO

close(2V), **lseek**(2V), **open**(2V), **read**(2V), **getwd**(3), **malloc**(3V), **dir**(5)

NOTES

The **directory** library routines now use a new include file, **<dirent.h>**. This replaces the file, **<sys/dir.h>**, used in previous releases. Furthermore, with the use of this new file, the **readdir( )** routine returns directory entries whose structure is named **struct dirent** rather than **struct direct** as before. The file **<sys/dir.h>** is retained in the current SunOS release for purposes of backwards source code compatibility; programs which use the **directory( )** library and **<sys/dir.h>** will continue to compile and run without source code modifications. However, existing programs should convert to the use of the new include file, **<dirent.h>**, as **<sys/dir.h>** will be removed in a future major release.

The *X/Open Portability Guide, issue 2* (XPG2) requires **<sys/dirent.h>** rather than **<dirent.h>**. **/usr/xpg2include/sys/dirent.h** is functionally equivalent to **/usr/include/dirent.h.** In future SunOS releases, X/Open conformance will require **<dirent.h>**.

NAME

dlopen, dlsym, dlerror, dlclose – simple programmatic interface to the dynamic linker

SYNOPSIS

#include <dlfcn.h>

void *dlopen(path, mode)
char *path; int mode;

void *dlsym(handle, symbol)
void *handle; char *symbol;

char *dlerror( )

int dlclose(handle);
void *handle;

DESCRIPTION

These functions provide a simple programmatic interface to the services of the dynamic link-editor. Operations are provided to add a new shared object to an program's address space, obtain the address bindings of symbols defined by such objects, and to remove such objects when their use is no longer required.

dlopen( ) provides access to the shared object in *path*, returning a descriptor that can be used for later references to the object in calls to **dlsym( )** and **dlclose( )**. If *path* was not in the address space prior to the call to **dlopen( )**, then it will be placed in the address space, and if it defines a function with the name *_init* that function will be called by **dlopen( )**. If, however, *path* has already been placed in the address space in a previous call to **dlopen( )**, then it will not be added a second time, although a count of **dlopen( )** operations on *path* will be maintained. *mode* is an integer containing flags describing options to be applied to the opening and loading process — it is reserved for future expansion and must always have the value 1. A null pointer supplied for *path* is interpreted as a reference to the "main" executable of the process. If **dlopen( )** fails, it will return a null pointer.

dlsym( ) returns the address binding of the symbol described in the null-terminated character string *symbol* as it occurs in the shared object identified by *handle*. The symbols exported by objects added to the address space by **dlopen( )** can be accessed *only* through calls to **dlsym( )**, such symbols do not supersede any definition of those symbols already present in the address space when the object is loaded, nor are they available to satisfy "normal" dynamic linking references. **dlsym( )** returns a null pointer if the symbol can not be found. A null pointer supplied as the value of *handle* is interpreted as a reference to the executable from which the call to **dlsym( )** is being made — thus a shared object can reference its own symbols.

dlerror returns a null-terminated character string describing the last error that occurred during a **dlopen( )**, **dlsym( )**, or **dlclose( )**. If no such error has occurred, then **dlerror( )** will return a null pointer. At each call to **dlerror( )**, the "last error" indication will be reset, thus in the case of two calls to **dlerror( )**, and where the second call follows the first immediately, the second call will always return a null pointer.

dlclose( ) deletes a reference to the shared object referenced by *handle*. If the reference count drops to 0, then if the object referenced by *handle* defines a function *_fini*, that function will be called, the object removed from the address space, and *handle* destroyed. If **dlclose( )** is successful, it will return a value of 0. A failing call to **dlclose( )** will return a non-zero value.

The object-intrinsic functions *_init* and *_fini* are called with no arguments and treated as though their types were **void**.

These functions are obtained by specifying **–ldl** as an option to **ld**(1).

SEE ALSO

ld(1), **link**(5)

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS

double drand48( )

double erand48(xsubi)
unsigned short xsubi[3];

long lrand48( )

long nrand48(xsubi)
unsigned short xsubi[3];

long mrand48( )

long jrand48(xsubi)
unsigned short xsubi[3];

void srand48(seedval)
long seedval;

unsigned short *seed48(seed16v)
unsigned short seed16v[3];

void lcong48(param)
unsigned short param[7];

DESCRIPTION

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

drand48( ) and **erand48( )** return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

lrand48( ) and **nrand48( )** return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

mrand48( ) and **jrand48( )** return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

srand48( ), **seed48( )**, and **lcong48( )** are initialization entry points, one of which should be invoked before either **drand48( )**, **lrand48( )**, or **mrand48( )** is called. Although it is not recommended practice, constant default initializer values will be supplied automatically if **drand48( )**, **lrand48( )**, or **mrand48( )** is called without a prior call to an initialization entry point. **erand48( )**, **nrand48( )**, and **jrand48( )** do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\mathrm{mod}\ m} \qquad n \geq 0.$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless **lcong48( )** has been invoked, the multiplier value a and the addend value c are given by

$$a = \mathrm{5DEECE66D}_{16} = 273673163155_8$$
$$c = \mathrm{B}_{16} = 13_8.$$

The value returned by any of the functions **drand48( )**, **erand48( )**, **lrand48( )**, **nrand48( )**, **mrand48( )**, or **jrand48( )** is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

drand48( ), **lrand48( )**, and **mrand48( )** store the last 48-bit X_i generated in an internal buffer; that is why they must be initialized prior to being invoked. The functions **erand48( )**, **nrand48( )**, and **jrand48( )** require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, functions **erand48( )**, **nrand48( )**, and **jrand48( )** allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, that is, the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function **srand48( )** sets the high-order 32 bits of X_i to the 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function **seed48( )** sets the value of X_i to the 48-bit value specified in the argument array. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by **seed48( )**, and a pointer to this buffer is the value returned by **seed48( )**. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last X_i value, and then use this value to reinitialize via **seed48( )** when the program is restarted.

The initialization function **lcong48( )** allows the user to specify the initial X_i, the multiplier value a, and the addend value c. Argument array elements *param*[0-2] specify X_i, *param*[3-5] specify the multiplier a, and *param*[6] specifies the 16-bit addend c. After **lcong48( )** has been called, a subsequent call to either **srand48( )** or **seed48( )** will restore the "standard" multiplier and addend values, a and c, specified on the previous page.

SEE ALSO

rand(3V)

NAME

econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, ecvt, fcvt, gcvt – output conversion

SYNOPSIS

```
#include <floatingpoint.h>

char *econvert(value, ndigit, decpt, sign, buf)
double value;
int ndigit, *decpt, *sign;
char *buf;

char *fconvert(value, ndigit, decpt, sign, buf)
double value;
int ndigit, *decpt, *sign;
char *buf;

char *gconvert(value, ndigit, trailing, buf)
double value;
int ndigit;
int trailing;
char *buf;

char *seconvert(value, ndigit, decpt, sign, buf)
single *value;
int ndigit, *decpt, *sign;
char *buf;

char *sfconvert(value, ndigit, decpt, sign, buf)
single *value;
int ndigit, *decpt, *sign;
char *buf;

char *sgconvert(value, ndigit, trailing, buf)
single *value;
int ndigit;
int trailing;
char *buf;

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

DESCRIPTION

econvert( ) converts the *value* to a null-terminated string of *ndigit* ASCII digits in *buf* and returns a pointer to *buf*. *buf* should contain at least *ndigit+1* characters. The position of the radix character relative to the beginning of the string is stored indirectly through *decpt*. Thus *buf* == "314" and **decpt* == 1 corresponds to the numerical value 3.14, while *buf* == "314" and **decpt* == –1 corresponds to the numerical value .0314. If the sign of the result is negative, the word pointed to by *sign* is nonzero; otherwise it is zero. The least significant digit is rounded.

fconvert works much like **econvert**, except that the correct digit has been rounded as if for **sprintf(%w.nf)** output with *n=ndigit* digits to the right of the radix character. *ndigit* can be negative to indicate rounding to the left of the radix character. The return value is a pointer to *buf*. *buf* should contain at least *310+max(0,ndigit)* characters to accomodate any double-precision *value*.

gconvert( ) converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf*. It produces *ndigit* significant digits in fixed-decimal format, like **sprintf(%w.nf)**, if possible, and otherwise in floating-decimal format, like **sprintf(%w.ne)**; in either case *buf* is ready for printing, with sign and exponent. The result corresponds to that obtained by

(void) sprintf(buf, "%w.ng", value);

If *trailing*= 0, trailing zeros and a trailing point are suppressed, as in **sprintf(%g)**. If *trailing*!= 0, trailing zeros and a trailing point are retained, as in **sprintf(%#g)**.

seconvert, **sfconvert**, and **sgconvert( )** are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C's usual conversion of single-precision arguments to double.

ecvt( ) and **fcvt( )** are obsolete versions of **econvert( )** and **fconvert( )** that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

gcvt( ) is an obsolete version of **gconvert( )** that always suppresses trailing zeros and point.

IEEE Infinities and NaNs are treated similarly by these functions. ''NaN'' is returned for NaN, and ''Inf'' or ''Infinity'' for Infinity. The longer form is produced when *ndigit* >= 8.

The radix character is determined by the current setting of the program's locale (category **LC_NUMERIC**). In the **"C"** locale or if the locale is undefined, the readix character defaults to a period '**.**'.

SEE ALSO

printf(3V)

NAME

end, etext, edata – last locations in program

SYNOPSIS

extern end;
extern etext;
extern edata;

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and **end**( ) above the uninitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with **end**, but it is reset by the routines **brk**(2), **malloc**(3V), standard input/output (**stdio**(3V)), the profile (**–p**) option of **cc**(1V), and so on. Thus, the current value of the program break should be determined by **sbrk(0)** (see **brk**(2)).

SEE ALSO

cc(1V), **brk**(2), **malloc**(3V), **stdio**(3V)

NAME

ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – Ethernet address mapping operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>

char *
ether_ntoa(e)
struct ether_addr *e;

struct ether_addr *ether_aton(s)
char *s;

ether_ntohost(hostname, e)
char *hostname;
struct ether_addr *e;

ether_hostton(hostname, e)
char *hostname;
struct ether_addr *e;

ether_line(l, e, hostname)
char *l;
struct ether_addr *e;
char *hostname;
```

DESCRIPTION

These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function **ether_ntoa**( ) converts a 48 bit Ethernet number pointed to by *e* to its standard ACSII representation; it returns a pointer to the ASCII string. The representation is of the form: *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff. The function **ether_aton**( ) converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.

The function **ether_ntohost**( ) maps an Ethernet number (pointed to by *e*) to its associated hostname. The string pointed to by **hostname** must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure. Inversely, the function **ether_hostton**( ) maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by *e*. The function also returns zero upon success and non-zero upon failure.

The function **ether_line**( ) scans a line (pointed to by *l*) and sets the hostname and the Ethernet number (pointed to by *e*). The string pointed to by **hostname** must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by **ethers**(5).

FILES

/etc/ethers (or the Network Information Service (NIS) maps **ethers.byaddr** and **ethers.byname**)

SEE ALSO

ethers(5)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

execl, execv, execle, execlp, execvp – execute a file

SYNOPSIS

int execl(path, arg0 [, arg1,... , argn] (char *)0)
char *path, *arg0, *arg1, ..., *argn;

int execv(path, argv)
char *path, *argv[ ];

int execle(path, arg0 [, arg1,... , argn] (char *)0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

int execlp(file, arg0 [, arg1,... , argn] (char *)0)
char *file, *arg0, *arg1, ..., *argn;

int execvp(file, argv)
char *file, *argv[ ];

extern char **environ;

DESCRIPTION

These routines provide various interfaces to the **execve( )** system call. Refer to **execve(2V)** for a description of their properties; only brief descriptions are provided here.

exec( ) in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful **exec( )**; the calling core image is lost.

The *filename* argument is a pointer to the name of the file to be executed. The pointers *arg*[0], *arg*[1]... address null-terminated strings. Conventionally *arg*[0] is the name of the file.

Two interfaces are available. **execl( )** is useful when a known file with known arguments is being called; the arguments to **execl( )** are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A **(char *)0** argument must end the argument list. The cast to type **char *** insures portability.

The **execv( )** version is useful when the number of arguments is unknown in advance; the arguments to **execv( )** are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

argv is directly usable in another **execv( )** because *argv*[*argc*] is 0.

envp is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an '=', and a null-terminated value. The array of pointers is terminated by a NULL pointer. The shell **sh**(1) passes an environment entry for each global shell variable defined when the program is called. See **environ**(5V) for some conventionally used names. The C run-time start-off routine places a copy of *envp* in the global cell *environ*, which is used by **execv( )** and **execl( )** to pass the environment to any subprograms executed by the current program.

execlp( ) and **execvp( )** are called with the same arguments as **execl( )** and **execv( )**, but duplicate the shell's actions in searching for an executable *file* in a list of directories. The directory list is obtained from the environment.

RETURN VALUES

These functions return to the calling process only on failure. They return −1 and set **errno** to indicate the error if *path* or *file* cannot be found, if it is not executable, if it does not start with a valid magic number (see **a.out**(5)), if maximum memory is exceeded, or if the arguments require too much space. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

ERRORS

If any of the following conditions occur, these functions will return and set **errno** to one of the following:

E2BIG — The number of bytes used by the new process image's argument list and environment list is greater than {ARG_MAX} bytes (see **sysconf**(2V)).

EACCES — Search permission is denied for a directory listed in the new process image file's path prefix.

The new process image file denies execution permission.

The new process image file is not a regular file.

ENAMETOOLONG — The length of the *path* or *file*, or an element of the environment variable **PATH** prefixed to a file, exceeds {PATH_MAX}.

A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect for that file (see **pathconf**(2V)).

ENOENT — One or more components of the new process image file's pathname do not exist.

ENOTDIR — A component of the new process image file's path prefix is not a directory.

if the following condition occurs, **execl**( ), **execv**( ), and **execle**( ) set **errno** to:

ENOEXEC — The new process image file has the appropriate access permission, but is not in the proper format.

If the following condition is detected, the exec functions set **errno** to:

ENOMEM — The new process image requires more memory than there is swap space available.

On Sun-3 systems, the new process image requires more than 2^{31} bytes.

SYSTEM V ERRORS

In addition to the above, if the following condition occurs, the exec functions set **errno** to:

ENOENT — *path* or *file* points to a null pathname.

FILES

/usr/bin/sh — shell, invoked if command *file* found by **execlp**( ) or **execvp**( )

SEE ALSO

csh(1), **sh**(1), **execve**(2V), **fork**(2V), **pathconf**(2V), **sysconf**(2V), **a.out**(5), **environ**(5V)

Programming Utilities and Libraries

NAME

exit – terminate a process after performing cleanup

SYNOPSIS

void
exit(status)
int status;

DESCRIPTION

exit( ) terminates a process by calling **exit**(2V) after calling any termination handlers named by calls to **on_exit**. Normally, this is just the Standard I/O library function **_cleanup**. **exit( )** never returns.

SEE ALSO

exit(2V), **intro**(3), **on_exit**(3)

NAME

exportent, getexportent, setexportent, addexportent, remexportent, endexportent, getexportopt – get exported file system information

SYNOPSIS

```
#include <stdio.h>
#include <exportent.h>

FILE *setexportent()

struct exportent *getexportent(filep)
FILE *filep;

int addexportent(filep, dirname, options)
FILE *filep;
char *dirname;
char *options;

int remexportent(filep, dirname)
FILE *filep;
char *dirname;

char *getexportopt(xent, opt)
struct exportent *xent;
char *opt;

void endexportent(filep)
FILE *filep;
```

DESCRIPTION

These routines access the exported filesystem information in **/etc/xtab**.

setexportent() opens the export information file and returns a file pointer to use with **getexportent**, **addexportent**, **remexportent**, and **endexportent**. **getexportent()** reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the file, **/etc/xtab** The fields have meanings described in **exports**(5).

```
#define ACCESS_OPT  "access"  /* machines that can mount fs */
#define ROOT_OPT    "root"    /* machines with root access of fs */
#define RO_OPT      "ro"      /* export read-only */
#define ANON_OPT    "anon"    /* uid for anonymous requests */
#define SECURE_OPT  "secure"  /* require secure NFS for access */
#define WINDOW_OPT  "window"  /* expiration window for credential */

struct exportent {
        char *xent_dirname;     /* directory (or file) to export */
        char *xent_options;     /* options, as above */
};
```

addexportent() adds the **exportent()** to the end of the open file *filep*. It returns 0 if successful and −1 on failure. **remexportent()** removes the indicated entry from the list. It also returns 0 on success and −1 on failure. **getexportopt()** scans the *xent_options* field of the **exportent()** structure for a substring that matches *opt*. It returns the string value of *opt*, or NULL if the option is not found.

endexportent() closes the file.

FILES

/etc/exports
/etc/xtab

SEE ALSO

exports(5), **exportfs**(8)

DIAGNOSTICS

NULL pointer (0) returned on EOF or error.

BUGS

The returned **exportent( )** structure points to static information that is overwritten in each call.

NAME

fclose, fflush – close or flush a stream

SYNOPSIS

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

DESCRIPTION

fclose( ) writes out any buffered data for the named stream, and closes the named stream. Buffers allocated by the standard input/output system are freed.

fclose( ) is performed automatically for all open files upon calling **exit**(3).

fflush( ) writes any unwritten data for an output stream or an update stream in which the most recent operation was not input to be delivered to the host environment to the file; otherwise it is ignored. The named stream remains open.

SYSTEM V DESCRIPTION

When **fflush( )** is called on a stream opened for reading, any unread data buffered in the stream is invalidated. When **fflush( )** is called on a stream opened for reading, if the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description is adjusted so the next operation on the open file description deals with the byte after the last byte read from or written to the stream being flushed.

RETURN VALUES

fclose( ) and **fflush( )** return:

0	on success.
EOF	if any error (such as trying to write to a file that has not been opened for writing) was detected.

SEE ALSO

close(2V), **exit**(3), **fopen**(3V), **setbuf**(3V)

NAME

ferror, feof, clearerr, fileno – stream status inquiries

SYNOPSIS

#include <stdio.h>

ferror(stream)
FILE *stream;

feof(stream)
FILE *stream;

clearerr(stream)
FILE *stream;

fileno(stream)
FILE *stream;

DESCRIPTION

ferror( ) returns non-zero when an error has occurred reading from or writing to the named stream, otherwise zero. Unless cleared by **clearerr( )**, the error indication lasts until the stream is closed.

feof( ) returns non-zero when EOF has previously been detected reading the named input stream, otherwise zero. Unless cleared by **clearerr( )**, the EOF indication lasts until the stream is closed.

clearerr( ) resets the error indication and EOF indication to zero on the named stream.

fileno( ) returns the integer file descriptor associated with the stream (see **open**(2V)).

SYSTEM V DESCRIPTION

feof( ) returns non-zero when EOF has previously been detected reading the named input stream, otherwise zero. Unless cleared by **clearerr( )**, the EOF indication lasts until the stream is closed, however, operations which attempt to read from the stream will ignore the current state of the EOF indication and attempt to read from the file descriptor associated with the stream.

SEE ALSO

open(2V), **fopen**(3V)

NOTES

These functions are defined in the C library and are also defined as macros in **<stdio.h>**.

NAME

single_to_decimal, double_to_decimal, extended_to_decimal – convert floating-point value to decimal record

SYNOPSIS

```
#include <floatingpoint.h>

void single_to_decimal(px, pm, pd, ps)
single *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void double_to_decimal(px, pm, pd, ps)
double *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void extended_to_decimal(px, pm, pd, ps)
extended *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;
```

DESCRIPTION

The **floating_to_decimal( )** functions convert the floating-point value at **px* into a decimal record at **pd*, observing the modes specified in **pm* and setting exceptions in **ps*. If there are no IEEE exceptions, **ps* will be zero.

If **px* is zero, infinity, or NaN, then only *pd->sign* and *pd->fpclass* are set. Otherwise *pd->exponent* and *pd->ds* are also set so that

(pd->sign)*(pd->ds)*10(pd->exponent)**

is a correctly rounded approximation to **px*. *pd->ds* has at least one and no more than **DECIMAL_STRING_LENGTH–1** significant digits because one character is used to terminate the string with a null character.

pd->ds is correctly rounded according to the IEEE rounding modes in *pm->rd*. **ps* has *fp_inexact* set if the result was inexact, and has *fp_overflow* set if the string result does not fit in *pd->ds* because of the limitation **DECIMAL_STRING_LENGTH.**

If *pm->df == floating_form*, then *pd->ds* always contains *pm->ndigits* significant digits. Thus if **px* == 12.34 and *pm->ndigits* == 8, then *pd->ds* will contain 12340000 and *pd->exponent* will contain –6.

If *pm->df == fixed_form* and *pm->ndigits* >= 0, then *pd->ds* always contains *pm->ndigits* after the point and as many digits as necessary before the point. Since the latter is not known in advance, the total number of digits required is returned in *pd->ndigits*; if that number >= **DECIMAL_STRING_LENGTH**, then *ds* is undefined. *pd->exponent* always gets *–pm->ndigits*. Thus if **px* == 12.34 and *pm->ndigits* == 1, then *pd->ds* gets 123, *pd->exponent* gets –1, and *pd->ndigits* gets 3.

If *pm->df == fixed_form* and *pm->ndigits* < 0, then *pm->ds* always contains *–pm->ndigits* trailing zeros; in other words, rounding occurs *–pm->ndigits* to the left of the decimal point, but the digits rounded away are retained as zeros. The total number of digits required is in *pd->ndigits*. *pd->exponent* always gets 0. Thus if **px* == 12.34 and *pm->ndigits* == –1, then *pd->ds* gets 10, *pd->exponent* gets 0, and *pd->ndigits* gets 2.

pd->more is not used.

econvert( ), **fconvert( )** and **gconvert( )** (see **econvert**(3)), and **printf( )** and **sprintf( )** (see **printf**(3V)) all use **double_to_decimal( )**.

SEE ALSO

econvert(3), **printf**(3V)

NAME

floatingpoint – IEEE floating point definitions

SYNOPSIS

#include <sys/ieeefp.h>
#include <floatingpoint.h>

DESCRIPTION

This file defines constants, types, variables, and functions used to implement standard floating point according to ANSI/IEEE Std 754-1985. The variables and functions are implemented in **libc.a**. The included file **<sys/ieeefp.h>** defines certain types of interest to the kernel.

IEEE Rounding Modes:

fp_direction_type The type of the IEEE rounding direction mode. Note: the order of enumeration varies according to hardware.

fp_direction The IEEE rounding direction mode currently in force. This is a global variable that is intended to reflect the hardware state, so it should only be written indirectly through a function like **ieee_flags ("set","direction",...)** that also sets the hardware state.

fp_precision_type The type of the IEEE rounding precision mode, which only applies on systems that support extended precision such as Sun-3 systems with 68881's.

fp_precision The IEEE rounding precision mode currently in force. This is a global variable that is intended to reflect the hardware state on systems with extended precision, so it should only be written indirectly through a function like **ieee_flags("set","precision",...)**.

SIGFPE handling:

sigfpe_code_type The type of a SIGFPE code.

sigfpe_handler_type The type of a user-definable SIGFPE exception handler called to handle a particular SIGFPE code.

SIGFPE_DEFAULT A macro indicating the default SIGFPE exception handling, namely to perform the exception handling specified by calls to **ieee_handler(3M)**, if any, and otherwise to dump core using **abort**(3).

SIGFPE_IGNORE A macro indicating an alternate SIGFPE exception handling, namely to ignore and continue execution.

SIGFPE_ABORT A macro indicating an alternate SIGFPE exception handling, namely to abort with a core dump.

IEEE Exception Handling:

N_IEEE_EXCEPTION The number of distinct IEEE floating-point exceptions.

fp_exception_type The type of the N_IEEE_EXCEPTION exceptions. Each exception is given a bit number.

fp_exception_field_type
The type intended to hold at least N_IEEE_EXCEPTION bits corresponding to the IEEE exceptions numbered by **fp_exception_type**. Thus **fp_inexact** corresponds to the least significant bit and **fp_invalid** to the fifth least significant bit. Note: some operations may set more than one exception.

fp_accrued_exceptions
The IEEE exceptions between the time this global variable was last cleared, and the last time a function like **ieee_flags("get","exception",...)** was called to update the variable by obtaining the hardware state.

ieee_handlers An array of user-specifiable signal handlers for use by the standard **SIGFPE** handler for IEEE arithmetic-related **SIGFPE** codes. Since IEEE trapping modes correspond to hardware modes, elements of this array should only be modified with a function like **ieee_handler**(3M) that performs the appropriate hardware mode update. If no **sigfpe_handler** has been declared for a particular IEEE-related **SIGFPE** code, then the related **ieee_handlers** will be invoked.

IEEE Formats and Classification:

single;*extended* Definitions of IEEE formats.

fp_class_type An enumeration of the various classes of IEEE values and symbols.

IEEE Base Conversion:

The functions described under **floating_to_decimal**(3) and **decimal_to_floating**(3) not only satisfy the IEEE Standard, but also the stricter requirements of correct rounding for all arguments.

DECIMAL_STRING_LENGTH
The length of a **decimal_string**.

decimal_string The digit buffer in a **decimal_record**.

decimal_record The canonical form for representing an unpacked decimal floating-point number.

decimal_form The type used to specify fixed or floating binary to decimal conversion.

decimal_mode A struct that contains specifications for conversion between binary and decimal.

decimal_string_form An enumeration of possible valid character strings representing floating-point numbers, infinities, or NaNs.

SEE ALSO

abort(3), **decimal_to_floating**(3), **econvert**(3), **floating_to_decimal**(3), **ieee_flags**(3M), **ieee_handler**(3M), **sigfpe**(3), **string_to_decimal**(3), **strtod**(3)

NAME

fopen, freopen, fdopen – open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fd, type)
int fd;
char *type;

DESCRIPTION

fopen() opens the file named by *filename* and associates a stream with it. If the open succeeds, **fopen()** returns a pointer to be used to identify the stream in subsequent operations.

filename points to a character string that contains the name of the file to be opened.

type is a character string having one of the following values:

r	open for reading
w	truncate or create for writing
a	append: open for writing at end of file, or create for writing
r+	open for update (reading and writing)
w+	truncate or create for update
a+	append; open or create for update at EOF

freopen() opens the file named by *filename* and associates the stream pointed to by *stream* with it. The *type* argument is used just as in **fopen**. The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, **freopen()** returns the original value of *stream*.

freopen() is typically used to attach the preopened streams associated with **stdin**, **stdout**, and **stderr** to other files.

fdopen() associates a stream with the file descriptor *fd*. File descriptors are obtained from calls like **open**(2V), **dup**(2V), **creat**(2V), or **pipe**(2V), which open files but do not return streams. Streams are necessary input for many of the Section 3S library routines. The *type* of the stream must agree with the access permissions of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek**(3S) or **rewind()**, and input may not be directly followed by output without an intervening **fseek()**, **rewind()**, or an input operation which encounters EOF.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek()** or **rewind()**, and input may not be directly followed by output without an intervening **fseek()**, **rewind()**, or an input operation which encounters end-of-file.

SYSTEM V DESCRIPTION

When a file is opened for append (that is, when *type* is **a** or **a+**), it is impossible to overwrite information already in the file. **fseek()** may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

RETURN VALUES

On success, **fopen()**, **freopen()**, and **fdopen()** return a pointer to **FILE** which identifies the opened stream. On failure, they return NULL.

SEE ALSO

open(2V), **pipe**(2V), **fclose**(3V), **fseek**(3S)

BUGS

In order to support the same number of open files that the system does, **fopen()** must allocate additional memory for data structures using **calloc()** after 64 files have been opened. This confuses some programs which use their own memory allocators.

NAME

fread, fwrite – buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

int fread (ptr, size, nitems, stream)
char *ptr;
int size;
int nitems;
FILE *stream;

int fwrite (ptr, size, nitems, stream)
char *ptr;
int size;
int nitems;
FILE *stream;
```

DESCRIPTION

fread( ) reads, into a block pointed to by *ptr*, *nitems* items of data from the named input stream *stream*, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. It returns the number of items actually read. **fread( )** stops reading if an end-of-file or error condition is encountered while reading from *stream*, or if *nitems* items have been read. **fread( )** leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. **fread( )** does not change the contents of the file referred to by *stream* .

fwrite( ) writes at most *nitems* items of data from the block pointed to by *ptr* to the named output stream *stream*. It returns the number of items actually written. **fwrite( )** stops writing when it has written *nitems* items of data or if an error condition is encountered on *stream*. **fwrite( )** does not change the contents of the block pointed to by *ptr*.

If *size* or *nitems* is non-positive, no characters are read or written and 0 is returned by both **fread( )** and **fwrite( )**.

SEE ALSO

read(2V), **write**(2V), **fopen**(3V), **getc**(3V), **gets**(3S), **putc**(3S), **puts**(3S), **printf**(3V), **scanf**(3V)

DIAGNOSTICS

fread( ) and **fwrite( )** return 0 upon end of file or error.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)
FILE *stream;

DESCRIPTION

fseek() sets the position of the next input or output operation on the stream. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

rewind(*stream***)** is equivalent to **fseek(***stream*, 0L, 0), except that no value is returned.

fseek() and **rewind()** undo any effects of **ungetc**(3S).

After **fseek()** or **rewind()**, the next operation on a file opened for update may be either input or output.

ftell() returns the offset of the current byte relative to the beginning of the file associated with the named stream.

SEE ALSO

lseek(2V), **fopen**(3V), **popen**(3S), **ungetc**(3S)

DIAGNOSTICS

fseek() returns −1 for improper seeks, otherwise zero. An improper seek can be, for example, an **fseek()** done on a file associated with a non-seekable device, such as a tty or a pipe; in particular, **fseek()** may not be used on a terminal, or on a file opened using **popen**(3S).

WARNING

Although on the UNIX system an offset returned by **ftell()** is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to a (non-UNIX) system requires that an offset be used by **fseek()** directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

NAME

ftok – standard interprocess communication package

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;

DESCRIPTION

All interprocess communication facilities require the user to supply a key to be used by the **msgget**(2), **semget**(2), and **shmget**(2) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the **ftok**( ) subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ftok( ) returns a key based on *path* and ID that is usable in subsequent **msgget**, **semget**, and **shmget**( ) system calls. *path* must be the path name of an existing file that is accessible to the process. ID is a character which uniquely identifies a project. Note: **ftok**( ) will return the same key for linked files when called with the same ID and that it will return different keys when called with the same file name but different IDs.

SEE ALSO

intro(2), **msgget**(2), **semget**(2), **shmget**(2)

DIAGNOSTICS

ftok( ) returns **(key_t) –1** if *path* does not exist or if it is not accessible to the process.

WARNING

If the file whose *path* is passed to **ftok**( ) is removed when keys still refer to the file, future calls to **ftok**( ) with the same *path* and ID will return an error. If the same file is recreated, then **ftok**( ) is likely to return a different key than it did the original time it was called.

NAME

ftw – walk a file tree

SYNOPSIS

```
#include <ftw.h>

int ftw(path, fn, depth)
char *path;
int (*fn)();
int depth;
```

DESCRIPTION

ftw() recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, **ftw()** calls *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a **stat()** structure (see **stat**(2V)) containing information about the object, and an integer. Possible values of the integer, defined in the **<ftw.h>** header file, are FTW_F for a file, FTW_D for a directory, FTW_DNR for a directory that cannot be read, and FTW_NS for an object for which **stat()** could not successfully be executed. If the integer is FTW_DNR, descendants of that directory will not be processed. If the integer is FTW_NS, the **stat()** structure will contain garbage. An example of an object that would cause FTW_NS to be passed to *fn* would be a file in a directory with read but without execute (search) permission.

ftw() visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within **ftw()** (such as an I/O error). If the tree is exhausted, **ftw()** returns zero. If *fn* returns a nonzero value, **ftw()** stops its tree traversal and returns whatever value was returned by *fn*. If **ftw()** detects an error, it returns −1, and sets the error type in **errno**.

ftw() uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. **ftw()** will run more quickly if *depth* is at least as large as the number of levels in the tree.

SEE ALSO

stat(2V), **malloc**(3V)

BUGS

Because **ftw()** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

It could be made to run faster and use less storage on deep structures at the cost of considerable complexity.

ftw() uses **malloc**(3V) to allocate dynamic storage during its operation. If **ftw()** is forcibly terminated, such as by **longjmp()** being executed by *fn* or an interrupt routine, **ftw()** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

NAME

getacinfo, getacdir, getacflg, getacmin, setac, endac – get audit control file information

SYNOPSIS

```
int getacdir(dir, len)
char *dir;
int len;

int getacmin(min_val)
int *min_val;

int getacflg(auditstring, len)
char *auditstring;
int len;

void setac( )

void endac( )
```

DESCRIPTION

When first called, **getacdir( )** provides information about the first audit directory in the **audit_control** file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in **audit_control**(5) The parameter *len* specifies the length of the buffer *dir* . On return, *dir* points to the directory entry.

getacmin( ) reads the minimum value from the **audit_control** file and returns the value in **min_val.** The minimum value specifies how full the file system to which the audit files are being written can get before the script **audit_warn** is invoked.

getacflg( ) reads the system audit value from the **audit_control** file and returns the value in *auditstring*. The parameter *len* specifies the length of the buffer *auditstring*.

Calling *setac* rewinds the **audit_control** file to allow repeated searches.

Calling *endac* closes the **audit_control** file when processing is complete.

RETURN VALUES

getacdir( ), **getacflg( )** and **getacmin( )** return:

0 on success.

–2 on failure and set **errno** to indicate the error.

getacmin( ) and **getacflg( )** return:

1 on EOF.

getacdir( ) returns:

–1 on EOF.

2 if the directory search had to start from the beginning because one of the other functions was called between calls to **getacdir( )**.

These functions return:

–3 if the directory entry format in the **audit_control** file is incorrect.

getacdir( ) and **getacflg( )** return:

–3 if the input buffer is too short to accommodate the record.

SEE ALSO

audit_control(5)

NAME

getauditflagsbin, getauditflagschar – convert audit flag specifications

SYNOPSIS

```
#include <sys/label.h>
#include <sys/audit.h>
#include <sys/auevents.h>

int getauditflagsbin(auditstring, masks)
char *auditstring;
audit_state_t *masks;

int getauditflagschar(auditstring, masks, verbose)
char *auditstring;
audit_state_t *masks;
int verbose;
```

DESCRIPTION

getauditflagsbin() converts the character representation of audit values pointed to by *auditstring* into **audit_state_t** fields pointed to by *masks*. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in **audit_control**(5).

getauditflagschar() converts the **audit_state_t** fields pointed to by *masks* into a string pointed to by *auditstring*. If *verbose* is zero, the short (2-character) flag names are used. If *verbose* is non-zero, the long flag names are used. *auditstring* should be large enough to contain the ASCII representation of the events.

auditstring contains a series of event names, each one identifying a single audit class, separated by commas. The **audit_state_t** fields pointed to by *masks* correspond to binary values defined in *audit.h*.

DIAGNOSTICS

−1 is returned on error and 0 on success.

SEE ALSO

audit.log(5), **audit_control**(5)

BUGS

This is not a very extensible interface.

NAME

getc, getchar, fgetc, getw – get character or integer from stream

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar( )

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

DESCRIPTION

getc( ) returns the next character (that is, byte) from the named input stream, as an integer. It also moves the file pointer, if defined, ahead one character in stream. **getchar( )** is defined as **getc(stdin)**. **getc( )** and **getchar( )** are macros.

fgetc( ) behaves like **getc( )**, but is a function rather than a macro. **fgetc( )** runs more slowly than **getc( )**, but it takes less space per invocation and its name can be passed as an argument to a function.

getw( ) returns the next C **int** (*word*) from the named input stream. **getw( )** increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. **getw( )** assumes no special alignment in the file.

RETURN VALUES

On success, **getc( )**, **getchar( )** and **fgetc( )** return the next character from the named input stream as an integer. On failure, or on EOF, they return EOF. The EOF condition is remembered, even on a terminal, and all subsequent operations which attempt to read from the stream will return EOF until the condition is cleared with **clearerr( )** (see **ferror**(3V)).

getw( ) returns the next C **int** from the named input stream on success. On failure, or on EOF, it returns EOF, but since EOF is a valid integer, use **ferror**(3V) to detect **getw( )** errors.

SYSTEM V RETURN VALUES

On failure, or on EOF, these functions return EOF. The EOF condition is remembered, even on a terminal, however, operations which attempt to read from the stream will ignore the current state of the EOF indication and attempt to read from the file descriptor associated with the stream.

SEE ALSO

ferror(3V), **fopen**(3V), **fread**(3S), **gets**(3S), **putc**(3S), **scanf**(3V), **ungetc**(3S)

WARNINGS

If the integer value returned by **getc( )**, **getchar( )**, or **fgetc( )** is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

BUGS

Because it is implemented as a macro, **getc( )** treats a stream argument with side effects incorrectly. In particular, **getc(*f++)** does not work sensibly. **fgetc( )** should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw( )** are machine-dependent, and may not be readable using **getw( )** on a different processor.

NAME

getcwd – get pathname of current working directory

SYNOPSIS

```
char *getcwd(buf, size)
char *buf;
int size;
```

DESCRIPTION

getcwd() returns a pointer to the current directory pathname. The value of *size* must be at least two greater than the length of the pathname to be returned.

If *buf* is a NULL pointer, **getcwd()** will obtain *size* bytes of space using **malloc**(3V). In this case, the pointer returned by **getcwd()** may be used as the argument in a subsequent call to **free()**.

The function is implemented by using **popen**(3S) to pipe the output of the **pwd**(1) command into the specified string space.

RETURN VALUES

getcwd() returns a pointer to the current directory pathname on success. If *size* is not large enough, or if an error occurs in a lower-level function, **getcwd()** returns NULL and sets **errno** to indicate the error.

ERRORS

EINVAL *size* is less than or equal to zero.

ERANGE *size* is greater than zero, but is smaller than the length of the pathname plus 1.

If the following condition is detected, **getcwd()** sets **errno** to:

EACCES Read or search permission is denied for a component of the pathname.

EXAMPLES

```
char *cwd, *getcwd();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
        perror ("pwd");
        exit (1);
}
printf("%s\n", cwd);
```

SEE ALSO

pwd(1), **getwd**(3), **malloc**(3V), **popen**(3S)

BUGS

Since this function uses **popen()** to create a pipe to the **pwd** command, it is slower than **getwd()** and gives poorer error diagnostics. **getcwd()** is provided only for compatibility with other UNIX operating systems.

NAME

getenv – return value for environment name

SYNOPSIS

#include <stdlib.h>

char *getenv(name)
char *name;

DESCRIPTION

getenv( ) searches the environment list (see **environ**(5V)) for a string of the form *name=value*, and returns a pointer to the string *value* if such a string is present. Otherwise, **getenv( )** returns NULL.

RETURN VALUES

On success, **getenv( )** returns a pointer to a string containing the value for the specified *name*. If the specified *name* cannot be found, it returns NULL.

SEE ALSO

environ(5V), **execve**(2V), **putenv**(3)

NAME

getfauditflags – generates the process audit state

SYNOPSIS

#include <sys/types.h>
#include <sys/audit.h>
#include <sys/label.h>

void getfauditflags(usremasks, usrdmasks, lastmasks)
audit_state_t *usremasks;
audit_state_t *usrdmasks;
audit_state_t *lastmasks;

DESCRIPTION

getfauditflags generates the process audit state from the user audit value as input to **getfauditflags** and the system audit value as specified in the **audit_control** file. **getfauditflags** obtains the system audit value by calling **getacflg**. The user audit value, pointed to by *usremasks* and *usrdmasks* is passed into **getfauditflags.**

usremasks points to **audit_state_t** fields which contains two values. The first value defines which events are *always* to be audited when they succeed. The second value defines which events are always to be audited when they fail.

usrdmasks also points to **audit_state_t** fields which contains two values. The first value defines which events are *never* to be audited when they succeed. The second value defines which events are never to be audited when they fail.

The structures pointed to by *usremasks* and *usrdmasks* may be obtained from the **passwd.adjunct** file by calling **getpwaent( )** which returns a pointer to a strucure containing all **passwd.adjunct** fields for a user.

lastmasks points to **audit_state_t** as well. The first value defines which events are to be audited when they succeed and the second value defines which events are to be audited when they fail.

Both *usremasks* and *usrdmasks* override the values in the system audit values.

DIAGNOSTICS

-1 is returned on error and 0 on success.

SEE ALSO

getauditflags(3), **getacinfo**(3), **audit.log**(5), **audit_control**(5)

NAME

getfsent, getfsspec, getfsfile, getfstype, setfsent, endfsent – get file system descriptor file entry

SYNOPSIS

#include <fstab.h>

struct fstab *getfsent( )

struct fstab *getfsspec(spec)
char *spec;

struct fstab *getfsfile(file)
char *file;

struct fstab *getfstype(type)
char *type;

int setfsent( )

int endfsent( )

DESCRIPTION

These routines are included for compatibility with 4.2 BSD; they have been superseded by the **getmntent**(3) library routines.

getfsent, *getfsspec*, *getfstype*, and *getfsfile* each return a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, **<fstab.h>**.

```
struct fstab {
        char    *fs_spec;
        char    *fs_file;
        char    *fs_type;
        int     fs_freq;
        int     fs_passno;
};
```

The fields have meanings described in **fstab**(5).

getfsent( ) reads the next line of the file, opening the file if necessary.

getfsent( ) opens and rewinds the file.

endfsent closes the file.

getfsspec and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered. *getfstype* does likewise, matching on the file system type field.

FILES

/etc/fstab

SEE ALSO

fstab(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

The return value points to static information which is overwritten in each call.

NAME

getgraent, getgranam, setgraent, endgraent, fgetgraent – get group adjunct file entry

SYNOPSIS

```
#include <stdio.h>
#include <grpadj.h>

struct group_adjunct *getgraent( )

struct group_adjunct *getgranam(name)
char *name;

struct group_adjunct *fgetgraent(f)
FILE *f;

void setgraent( )

void endgraent( )
```

DESCRIPTION

getgraent( ) and **getgranam( )** each return pointers to an object with the following structure containing the broken-out fields of a line in the group adjunct file. Each line contains a **group_adjunct** structure, defined in the **<grpadj.h>** header file.

```
struct  group_adjunct {
        char   *gra_name;       /* the name of the group */
        char   *gra_passwd;     /* the encrypted group password */
};
```

When first called, **getgraent( )** returns a pointer to a **group_adjunct** structure corresponding to the first line in the file. Thereafter, it returns a pointer to the next **group_adjunct** structure in the file. So successive calls may be used to traverse the entire file.

For locating a particular group, **getgranam( )** searches through the file until it finds group *filename*, then returns a pointer to that structure.

A call to **getgraent( )** rewinds the group adjunct file to allow repeated searches. A call to **endgraent( )** closes the group adjunct file when processing is complete.

Because read access is required on **/etc/security/group.adjunct, getgraent( )** and **getgranam( )** will fail unless the calling process has effective UID of root.

FILES

/etc/security/group.adjunct
/var/yp/*domainname***/group.adjunct**

SEE ALSO

getlogin(3V), getgrent(3V), getpwaent(3), getpwent(3V), ypserv(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent – get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent( )

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

void setgrent( )

void endgrent( )

struct group *fgetgrent(f)
FILE *f;
```

DESCRIPTION

getgrent( ), **getgrgid( )** and **getgrnam( )** each return pointers to an object with the following structure containing the fields of a line in the group file. Each line contains a "group" structure, defined in **<grp.h>**.

```
struct  group {
        char    *gr_name;       /* name of the group */
        char    *gr_passwd;     /* encrypted password of the group */
        gid_t   gr_gid;         /* numerical group ID */
        char    **gr_mem;       /* null-terminated array of pointers to the
                                        individual member names */
};
```

getgrent( ) when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. **getgrgid( )** searches from the beginning of the file until a numerical group ID matching **gid** is found and returns a pointer to the particular structure in which it was found. **getgrnam( )** searches from the beginning of the file until a group name matching *name* is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **setgrent( )** has the effect of rewinding the group file to allow repeated searches. **endgrent( )** may be called to close the group file when processing is complete.

fgetgrent( ) returns a pointer to the next group structure in the stream *f*, which must refer to an open file in the same format as the group file **/etc/group**.

RETURN VALUES

getgrent( ), **getgrgid( )**, and **getgrnam( )** return a pointer to **struct group** on success. On EOF or error, they return NULL.

FILES

/etc/group

SEE ALSO

getlogin(3V), **getpwent**(3V), **group**(5), **ypserv**(8)

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

Unlike the corresponding routines for passwords (see **getpwent**(3v)), which always search the entire file, these routines start searching from the current file location.

WARNING

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.

NAME

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent – get network host entry

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct hostent *gethostent( )

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr;
int len, type;

sethostent(stayopen)
int stayopen
endhostent( )
```

DESCRIPTION

gethostent, **gethostbyname**, and **gethostbyaddr**( ) each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, **/etc/hosts**. In the case of **gethostbyaddr**( ), *addr* is a pointer to the binary format address of length *len* (not a character string).

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
};
```

The members of this structure are:

h_name — Official name of the host.

h_aliases — A zero terminated array of alternate names for the host.

h_addrtype — The type of address being returned; currently always **AF_INET**.

h_length — The length, in bytes, of the address.

h_addr_list — A pointer to a list of network addresses for the named host. Host addresses are returned in network byte order.

gethostent( ) reads the next line of the file, opening the file if necessary.

sethostent( ) opens and rewinds the file. If the *stayopen* flag is non-zero, the host data base will not be closed after each call to **gethostent**( ) (either directly, or indirectly through one of the other "gethost" calls).

endhostent( ) closes the file.

gethostbyname( ) and **gethostbyaddr**( ) sequentially search from the beginning of the file until a matching host name or host address is found, or until end-of-file is encountered. Host addresses are supplied in network order.

FILES

/etc/hosts

SEE ALSO

hosts(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

getlogin – get login name

SYNOPSIS

char *getlogin( )

DESCRIPTION

getlogin( ) returns a pointer to the login name as found in **/etc/utmp**. It may be used in conjunction with **getpwnam( )** to locate the correct password file entry when the same user ID is shared by several login names.

If **getlogin( )** is called within a process that is not attached to a terminal, or if there is no entry in **/etc/utmp** for the process's terminal, it returns a NULL pointer. The correct procedure for determining the login name is to call **cuserid( )**, or to call **getlogin( )** and, if it fails, to call **getpwuid(getuid( ))**.

FILES

/etc/utmp

SEE ALSO

cuserid(3v), **getpwent**(3v), **utmp**(5V)

RETURN VALUES

getlogin( ) returns a pointer to the login name on success. If the name is not found, it returns NULL.

BUGS

The return values point to static data whose content is overwritten by each call.

getlogin( ) does not work for processes running under a **pty** (for example, emacs shell buffers, or shell tools) unless the program ''fakes'' the login name in the **/etc/utmp** file.

NAME

getmntent, setmntent, addmntent, endmntent, hasmntopt – get file system descriptor file entry

SYNOPSIS

```
#include <stdio.h>
#include <mntent.h>

FILE *setmntent(filep, type)
char *filep;
char *type;

struct mntent *getmntent(filep)
FILE *filep;

int addmntent(filep, mnt)
FILE *filep;
struct mntent *mnt;

char *hasmntopt(mnt, opt)
struct mntent *mnt;
char *opt;

int endmntent(filep)
FILE *filep;
```

DESCRIPTION

These routines replace the **getfsent( )** routines for accessing the file system description file **/etc/fstab**. They are also used to access the mounted file system description file **/etc/mtab**.

setmntent( ) opens a file system description file and returns a file pointer which can then be used with **getmntent, addmntent**, or **endmntent**. The *type* argument is the same as in **fopen**(3V). **getmntent( )** reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, **<mntent.h>**. On failure, **getmntent( )** returns the NULL pointer. The fields have meanings described in **fstab**(5).

```
struct mntent{
        char   *mnt_fsname;  /* name of mounted file system */
        char   *mnt_dir;     /* file system path prefix */
        char   *mnt_type;    /* MNTTYPE_* */
        char   *mnt_opts;    /* MNTOPT* */
        int    mnt_freq;     /* dump frequency, in days */
        int    mnt_passno;   /* pass number on parallel fsck */
};
```

addmntent( ) adds the **mntent** structure *mnt* to the end of the open file *filep*. **addmntent( )** returns 0 on success, 1 on failure. Note: *filep* has to be opened for writing if this is to work. **hasmntopt( )** scans the **mnt_opts** field of the **mntent** structure *mnt* for a substring that matches *opt*. It returns the address of the substring if a match is found, 0 otherwise. **endmntent( )** closes the file. It always returns 1, so should be treated as type **void.**

FILES

/etc/fstab
/etc/mtab

SEE ALSO

fopen(3V), **getfsent**(3), **fstab**(5)

DIAGNOSTICS

NULL pointer (0) returned on EOF or error.

BUGS

The returned **mntent** structure points to static information that is overwritten in each call.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent( )

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net, type)
long net;
int type;

setnetent (stayopen)
int stayopen;

endnetent( )
```

DESCRIPTION

getnetent, **getnetbyname**, and **getnetbyaddr( )** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, **/etc/networks**.

```
struct  netent {
        char    *n_name;        /* official name of net */
        char    **n_aliases;    /* alias list */
        int     n_addrtype;     /* net number type */
        long    n_net;          /* net number */
};
```

The members of this structure are:

n_name The official name of the network.

n_aliases A zero terminated list of alternate names for the network.

n_addrtype The type of the network number returned; currently only AF_INET.

n_net The network number. Network numbers are returned in machine byte order.

getnetent( ) reads the next line of the file, opening the file if necessary.

setnetent( ) opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **setnetent( )** (either directly, or indirectly through one of the other “getnet” calls).

endnetent( ) closes the file.

getnetbyname( ) and **getnetbyaddr( )** sequentially search from the beginning of the file until a matching net name or net address and type is found, or until end-of-file is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

Only Internet network numbers are currently understood.

NAME

getnetgrent, setnetgrent, endnetgrent, innetgr – get network group entry

SYNOPSIS

getnetgrent(machinep, userp, domainp)
char **machinep, **userp, **domainp;

setnetgrent(netgroup)
char *netgroup

endnetgrent( )

innetgr(netgroup, machine, user, domain)
char *netgroup, *machine, *user, *domain;

DESCRIPTION

getnetgrent( ) returns the next member of a network group. After the call, *machinep* will contain a pointer to a string containing the name of the machine part of the network group member, and similarly for *userp* and *domainp*. If any of *machinep*, *userp* or *domainp* is returned as a NULL pointer, it signifies a wild card. **getnetgrent( )** will use **malloc**(3V) to allocate space for the name. This space is released when a **endnetgrent( )** call is made. **getnetgrent( )** returns 1 if it succeeded in obtaining another member of the network group, 0 if it has reached the end of the group.

getnetgrent( ) establishes the network group from which **getnetgrent( )** will obtain members, and also restarts calls to **getnetgrent( )** from the beginning of the list. If the previous **setnetgrent( )** call was to a different network group, a **endnetgrent( )** call is implied. **endnetgrent( )** frees the space allocated during the **getnetgrent( )** calls. **innetgr** returns 1 or 0, depending on whether *netgroup* contains the machine, user, domain triple as a member. Any of the three strings *machine*, *user*, or *domain* can be NULL, in which case it signifies a wild card.

FILES

/etc/netgroup

WARNINGS

The Network Information Service (NIS) must be running when using **getnetgrent( ),** since it only inspects the NIS netgroup map, never the local files.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

getopt, optarg, optind – get option letter from argument vector

SYNOPSIS

```
int getopt(argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind, opterr;
```

DESCRIPTION

getopt() returns the next option letter in *argv* that matches a letter in *optstring*. *optstring* must contain the option letters the command using **getopt()** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

optarg is set to point to the start of the option argument on return from **getopt**.

getopt() places in **optind** the *argv* index of the next argument to be processed. **optind** is external and is initialized to **1** before the first call to **getopt**.

When all options have been processed (that is, up to the first non-option argument), **getopt()** returns **−1**. The special option "—" may be used to delimit the end of the options; when it is encountered, **−1** will be returned, and "—" will be skipped.

DIAGNOSTICS

getopt() prints an error message on the standard error and returns a question mark (**?**) when it encounters an option letter not included in *optstring* or no option-argument after an option that expects one. This error message may be disabled by setting **opterr** to **0**.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an option argument:

```
main(argc, argv)
int argc;
char **argv;
{
        int c;
        extern char *optarg;
        extern int optind;
        .
        .
        .
        while ((c = getopt(argc, argv, "abo:")) != -1)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc ();
                        break;
```

```
                case 'o':
                        ofile = optarg;
                        break;
                case '?':
                        errflg++;
                }
        if (errflg) {
                (void)fprintf(stderr, "usage: . . . ");
                exit (2);
        }
        for (; optind < argc; optind++) {
                if (access(argv[optind], 4)) {
        .
        .
        .
}
```

SEE ALSO

getopts(1)

WARNING

Changing the value of the variable **optind**, or calling **getopt**( ) with different values of *argv*, may lead to unexpected results.

NAME

getpass – read a password

SYNOPSIS

char *getpass(prompt)
char *prompt;

DESCRIPTION

getpass( ) reads up to a NEWLINE or EOF from the file **/dev/tty**, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

SYSTEM V DESCRIPTION

If **/dev/tty** cannot be opened, **getpass( )** returns a NULL pointer. It does not read the standard input.

FILES

/dev/tty

SEE ALSO

crypt(3)

NOTES

The above routine uses **<stdio.h>**, which increases the size of programs not otherwise using standard I/O, more than might be expected.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS

#include <netdb.h>

struct protoent *getprotoent( )

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

setprotoent(stayopen)
int stayopen;

endprotoent( )

DESCRIPTION

getprotoent, **getprotobyname**, and **getprotobynumber( )** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, **/etc/protocols.**

```
struct  protoent {
        char    *p_name;        /* official name of protocol */
        char    **p_aliases;    /* alias list */
        int     p_proto;        /* protocol number */
};
```

The members of this structure are:

p_name	The official name of the protocol.
p_aliases	A zero terminated list of alternate names for the protocol.
p_proto	The protocol number.

getprotoent( ) reads the next line of the file, opening the file if necessary.

setprotoent( ) opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getprotoent( )** (either directly, or indirectly through one of the other "getproto" calls).

endprotoent( ) closes the file.

getprotobyname( ) and **getprotobynumber( )** sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until end-of-file is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getpw – get name from uid

SYNOPSIS

getpw(uid, buf)
char *buf;

DESCRIPTION

getpw( ) is obsoleted by **getpwent**(3V).

getpw( ) searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3V), **passwd**(5)

DIAGNOSTICS

Non-zero return on error.

NAME

getpwaent, getpwanam, setpwaent, endpwaent, fgetpwaent – get password adjunct file entry

SYNOPSIS

```
#include <sys/types.h>
#include <sys/label.h>
#include <sys/audit.h>
#include <pwdadj.h>

struct passwd_adjunct *getpwaent( )

struct passwd_adjunct *getpwanam(name)
char *name;

struct passwd_adjunct *fgetpwaent(f)
FILE *f;

void setpwaent( )

void endpwaent( )
```

DESCRIPTION

Both **getpwaent( )** and **getpwanam( )** return a pointer to an object with the following structure containing the broken-out fields of a line in the password adjunct file. Each line in the file contains a **passwd_adjunct** structure, declared in the **<pwdadj.h>** header file:

```
struct  passwd_adjunct {
        char          *pwa_name;
        char          *pwa_passwd;
        blabel_t      pwa_minimum;
        blabel_t      pwa_maximum;
        blabel_t      pwa_def;
        audit_state_t pwa_au_always;
        audit_state_t pwa_au_never;
        int           pwa_version;
};
```

When first called, **getpwaent( )** returns a pointer to a **passwd_adjunct** structure describing data from the first line in the file. Thereafter, it returns a pointer to a **passwd_adjunct** structure describing data from the next line in the file. So successive calls can be used to search the entire file.

getpwanam( ) searches from the beginning of the file until it finds a login name matching *name*, then returns a pointer to the particular structure in which it was found.

Calling **setpwaent( )** rewinds the password adjunct file to allow repeated searches. Calling **endpwaent( )** closes the password adjunct file when processing is complete.

Because read access is required on **/etc/security/passwd.adjunct, getpwaent( )** and **getpwanam( )** will fail unless the calling process has effective UID of root.

FILES

/etc/security/passwd.adjunct
/var/yp/*domainname*/passwd.adjunct.byname

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

SEE ALSO

getpwent(3V), **getgrent**(3V), **passwd.adjunct**(5), **ypserv**(8)

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile, fgetpwent – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwent( )

struct passwd *getpwuid(uid)
uid_t uid;

struct passwd *getpwnam(name)
char *name;

void setpwent( )

void endpwent( )

int setpwfile(name)
char *name;

struct passwd *fgetpwent(f)
FILE *f;

DESCRIPTION

getpwent( ), **getpwuid( )** and **getpwnam( )** each return a pointer to an object with the following structure containing the fields of a line in the password file. Each line in the file contains a **passwd** structure, declared in the **<pwd.h>** header file:

```
struct  passwd {
        char    *pw_name;
        char    *pw_passwd;
        uid_t   pw_uid;
        gid_t   pw_gid;
        int     pw_quota;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
struct passwd *getpwent( ), *getpwuid( ), *getpwnam( );
```

The fields **pw_quota** and **pw_comment** are unused; the others have meanings described in **passwd**(5). When first called, **getpwent( )** returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. **getpwuid( )** searches from the beginning of the file until a numerical user ID matching *uid* is found and returns a pointer to the particular structure in which it was found. **getpwnam( )** searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **setpwent( )** has the effect of rewinding the password file to allow repeated searches. **endpwent( )** may be called to close the password file when processing is complete.

setpwfile( ) changes the default password file to *name* thus allowing alternate password files to be used. Note: it does *not* close the previous file. If this is desired, **endpwent( )** should be called prior to it. **setpwfile( )** will fail if it is called before a call to one of **getpwent( )**, **getpwuid( )**, **setpwent( )**, or **getpwnam( )** , or if it is called before a call to one of these functions and after a call to **endpwent( )**.

fgetpwent( ) returns a pointer to the next passwd structure in the stream *f*, which matches the format of the password file **/etc/passwd**.

SYSTEM V DESCRIPTION

struct passwd is declared in **pwd.h** as:

```
struct  passwd {
        char    *pw_name;
        char    *pw_passwd;
        uid_t   pw_uid;
        gid_t   pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

The field **pw_age** is used to hold a value for "password aging" on some systems; "password aging" is not supported on Sun systems.

RETURN VALUES

getpwent( ), **getpwuid( )**, and **getpwnam( )** return a pointer to **struct passwd** on success. On EOF or error, or if the requested entry is not found, they return NULL.

setpwfile( ) returns:

1 on success.

0 on failure.

FILES

/etc/passwd
/var/yp/*domainname***/passwd.byname**
/var/yp/*domainname***/passwd.byuid**

SEE ALSO

getgrent(3V), **issecure**(3), **getlogin**(3V), **passwd**(5), **ypserv**(8)

NOTES

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.

setpwfile( ) and **fgetpwent( )** are obsolete and should not be used, because when the system is running in secure mode (see **issecure**(3)), the password file only contains part of the information needed for a user database entry.

BUGS

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

NAME

getrpcent, getrpcbyname, getrpcbynumber, endrpcent, setrpcent – get RPC entry

SYNOPSIS

```
#include <netdb.h>

struct rpcent *getrpcent( )

struct rpcent *getrpcbyname(name)
char *name;

struct rpcent *getrpcbynumber(number)
int number;

setrpcent (stayopen)
int stayopen

endrpcent ()
```

DESCRIPTION

getrpcent, **getrpcbyname**, and **getrpcbynumber**( ) each return a pointer to an object with the following structure containing the broken-out fields of a line in the rpc program number data base, **/etc/rpc**.

```
struct  rpcent {
        char    *r_name;        /* name of server for this rpc program */
        char    **r_aliases;    /* alias list */
        long    r_number;       /* rpc program number */
};
```

The members of this structure are:

r_name	The name of the server for this rpc program.
r_aliases	A zero terminated list of alternate names for the rpc program.
r_number	The rpc program number for this service.

getrpcent( ) reads the next line of the file, opening the file if necessary.

setrpcent( ) opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getrpcent**( ) (either directly, or indirectly through one of the other "getrpc" calls).

endrpcent closes the file.

getrpcbyname( ) and **getrpcbynumber**( ) sequentially search from the beginning of the file until a matching rpc program name or program number is found, or until end-of-file is encountered.

FILES

/etc/rpc

SEE ALSO

rpc(5), **rpcinfo**(8C), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

gets, fgets – get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

DESCRIPTION

gets( ) reads characters from the standard input stream, **stdin**, into the array pointed to by *s*, until a NEWLINE character is read or an EOF condition is encountered. The NEWLINE character is discarded and the string is terminated with a null character. **gets( )** returns its argument.

fgets( ) reads characters from the stream into the array pointed to by *s*, until $n-1$ characters are read, a NEWLINE character is read and transferred to *s*, or an EOF condition is encountered. The string is then terminated with a null character. **fgets( )** returns its first argument.

SEE ALSO

puts(3S), **getc**(3V), **scanf**(3V), **fread**(3S), **ferror**(3V)

BUGS

If the input to **gets ( )** or **fgets ( )** contains a null character, the null terminates the input, and all subsequent data will be lost.

DIAGNOSTICS

If EOF is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a NULL pointer is returned. Otherwise *s* is returned.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

#include <netdb.h>

struct servent *getservent( )

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen;

endservent( )

DESCRIPTION

getservent, *getservbyname*, and *getservbyport* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, **/etc/services**.

```
struct   servent {
         char    *s_name;        /* official name of service */
         char    **s_aliases;    /* alias list */
         int     s_port;         /* port service resides at */
         char    *s_proto;       /* protocol to use */
};
```

The members of this structure are:

s_name	The official name of the service.
s_aliases	A zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network short byte order.
s_proto	The name of the protocol to use when contacting the service.

getservent( ) reads the next line of the file, opening the file if necessary.

getservent( ) opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getservent( )** (either directly, or indirectly through one of the other "getserv" calls).

endservent( ) closes the file.

getservbyname( ) and **getservbyport( )** sequentially search from the beginning of the file until a matching protocol name or port number is found, or until end-of-file is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3N), **services**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

getsubopt – parse sub options from a string.

SYNOPSIS

```
int getsubopt(optionp, tokens, valuep)
char    **optionp;
char    *tokens[];
char    **valuep;
```

DESCRIPTION

getsubopt() is a function to parse suboptions in a flag argument that was initially parsed by **getopt(3)**. These suboptions are separated by commas and may consist of either a single token, or a token-value pair separated by an equal sign. Since commas delimit suboptions in the option string they are not allowed to be part of the suboption or the value of a suboption. An example command that uses this syntax is **mount(8)**, which allows you to specify mount parameters with the *-o* switch as follows :

pepper % mount -o rw,hard,bg,wsize=1024 speed:/usr /usr

In this example there are four suboptions: 'rw', 'hard', 'bg', and 'wsize', the last of which has an associated value of 1024.

getsubopt() takes the address of a pointer to the option string, a vector of possible tokens, and the address of a value string pointer. It returns the index of the token that matched the suboption in the input string or -1 if there was no match. If the option string at **optionp* contains only one subobtion, **getsubopt()** updates **optionp* to point to the NUL at the end of the string, otherwise it isolates the suboption by replacing the comma seperator with a NUL, and updates **optionp* to point to the start of the next suboption. If the suboption has an associated value, **getsubopt()** updates **valuep* to point to the value's first character. Otherwise it sets **valuep* to NULL.

The token vector is organized as a series of pointers to null-terminated strings. The end of the token vector is identified by a NULL pointer.

When **getsubopt()** returns, if **valuep* is not NULL, then the suboption processed included a value. The calling program may use this information to determine if the presence or lack of a value for this subobtion is an error.

Additionally, when **getsubopt()** fails to match the suboption with the tokens in the *tokens* array, the calling program should decide if this is an error, or if the unrecognized option should be passed on to another program.

DIAGNOSTICS

getsubopt() returns -1 when the token it is scanning is not in the token vector. The variable addressed by *valuep* contains a pointer to the first character of the *token* that was not recognized rather than a pointer to a value for that token.

The variable addressed by *optionp* points to the next option to be parsed, or a NUL character if there are no more options.

EXAMPLE

The following code fragment shows how you might process options to the **mount(8)** command using **getsubopt(3)**.

```
char *myopts[] = {
#define READONLY        0
                        "ro",
#define READWRITE       1
                        "rw",
#define WRITESIZE       2
                        "wsize",
#define READSIZE        3
                        "rsize",
                        NULL };
```

```
main(argc, argv)
        int  argc;
        char **argv;
{
        int sc, c, errflag;
        char *options, *value;
        extern char *optarg;
        extern int optind;
        .
        .
        .
        while((c = getopt(argc, argv, "abf:o:")) != -1) {
                switch (c) {
                case 'a': /* process a option */
                        break;
                case 'b': /* process b option */
                        break;
                case 'f':
                        ofile = optarg;
                        break;
                case '?':
                        errflag++;
                        break;
                case 'o':
                        options = optarg;
                        while (*options != '\0') {
                                switch(getsubopt(&options,myopts,&value) {
                                case READONLY : /* process ro option */
                                        break;
                                case READWRITE : /* process rw option */
                                        break;
                                  case WRITESIZE : /* process wsize option */
                                        if (value == NULL) {
                                                error_no_arg();
                                                errflag++;
                                        } else
                                                write_size = atoi(value);
                                        break;
                                case READSIZE : /* process rsize option */
                                        if (value == NULL) {
                                                error_no_arg();
                                                errflag++;
                                        } else
                                                read_size = atoi(value);
                                        break;
                                default :
                                        /* process unknown token */
                                        error_bad_token(value);
                                        errflag++;
                                        break;
                                }
                        }
                        break;
```

```
                }
        }
        if (errflag) {
                /* print Usage instructions etc. */
        }
        for (; optind<argc; optind++) {
                /* process remaining arguments */
        }
        .
        .
        .
}
```

SEE ALSO

getopt(3)

NOTES

During parsing, commas in the option input string are changed to nulls.

White space in tokens or token-value pairs must be protected from the shell by quotes.

NAME

gettext, textdomain – retrieve a message string, get and set text domain

SYNOPSIS

char *gettext(msgtag)
char *msgtag;

char *textdomain(domainname)
char *domainname;

DESCRIPTION

gettext( ) returns a pointer to a null-terminated string (target string). *msgtag* is a string used at run-time to select the target string from the current domain of the active pool of messages. The length and contents of strings returned by **gettext( )** are undetermined until called at run-time. The string returned by **gettext( )** cannot be modified by the caller, but may be overwritten by a subsequent call to **gettext( )**. The **LC_MESSAGES** locale category setting determines the locale of strings that **gettext( )** returns.

The calling process can dynamically change the choice of locale for strings returned by **gettext( )** by invoking the **setlocale**(3V) function with the correct category and the required locale. If **setlocale( )** is not called or is called with an invalid value, **gettext( )** defaults to the **"C"** locale. The default name for the current domain is the empty string.

gettext( ) first attempts to resolve the target string from the active domain and locale of the message pool. The current locale and domain are determined by the combination of both the **LC_MESSAGES** category of locale and the current domain setting.

If the target string cannot be found by using the current locale and domain then *msgtag* and current domain are applied to the implementation-defined default locale (this default locale could contain any language). If the default locale does not also contain the target string then the *msgtag* and current domain will be applied to the **"C"** locale of the message pool. If the target string still cannot be found then **gettext( )** will return *msgtag*.

Any of the following conditions will result in a message not being found in the string archive:

- Non-existent archive selected after **setlocale( )** or **textdomain( )** was called.
- Non-existent archive in the **"C"** environment if **setlocale( )** was not called.
- Non-existent or deleted entry in the archive.

textdomain( ) sets the current domain to *domainname*. Subsequent calls to **gettext( )** refer to this domain. If *domainname* is NULL, **textdomain( )** returns the name of the current domain without changing it.

The setting of domain made by the last successful **textdomain( )** call remains valid across any number of subsequent calls to **setlocale( )**.

RETURN VALUES

gettext( ) returns a pointer to the null-terminated target string on success. On failure, **gettext( )** returns *msgtag*.

textdomain( ) returns a pointer to the name of the current domain. If the domain has not been set prior to this call, **textdomain( )** returns a pointer to an empty string. **textdomain( )** returns NULL if:

- *domainname* contains an invalid character.
- *domainname* is longer than **LINE_MAX** bytes in length.
- If, at the time of the call to **textdomain( )**, the combination of current locale and *domainname* creates a domain that does not exist at run-time. Note: in this case **textdomain( )** may have been called prior to a successful **setlocale**(3V) call, but **textdomain( )** will always check against current locale setting.

EXAMPLES

The following produces 'Hit Return\n' in a locale that is invalid or is valid and contains the same target string as the key:

```
printf( gettext( "Hit Return\n" );
```

On a system whose default language is French, and whose process has the LC_MESSAGES category validly set, the following might print: 'Bonjour':

```
setlocale( LC_MESSAGES, "" );
textdomain( "Morning" );
printf( gettext( "Welcome" );
```

If the LC_MESSAGES category was invalidly set and the default (**LC_DEFAULT**) is set to English, the last example above might print 'Good morning'. If the default is not set or is also invalid, the example would print 'Welcome'.

SEE ALSO

setlocale(3V), **installtxt**(8)

NAME

getttyent, getttynam, setttyent, endttyent – get ttytab file entry

SYNOPSIS

#include <ttyent.h>

struct ttyent *getttyent( )

struct ttyent *getttynam(name)
char *name;

setttyent( )

endttyent( )

DESCRIPTION

getttyent( ) and **getttynam( )** each return a pointer to an object with the following structure containing the broken-out fields of a line from the tty description file.

```
struct  ttyent {
        char    *ty_name;       /* terminal device name */
        char    *ty_getty;      /* command to execute, usually getty */
        char    *ty_type;       /* terminal type for termcap (3X) */
        int     ty_status;      /* status flags (see below for defines) */
        char    *ty_window;     /* command to start up window manager */
        char    *ty_comment;    /* usually the location of the terminal */
};
#define TTY_ON          0x1     /* enable logins (startup getty) */
#define TTY_SECURE      0x2     /* allow root to login */
```

ty_name is the name of the character-special file in the directory /**dev**. For various reasons, it must reside in the directory /**dev**.

ty_getty is the command (usually **getty**(8)) which is invoked by **init** to initialize tty line characteristics. In fact, any arbitrary command can be used; a typical use is to initiate a terminal emulator in a window system.

ty_type is the name of the default terminal type connected to this tty line. This is typically a name from the **termcap**(5) data base. The environment variable TERM is initialized with this name by **getty**(8) or **login**(1).

ty_status is a mask of bit fields which indicate various actions to be allowed on this tty line. The following is a description of each flag.

TTY_ON
Enables logins (that is, **init**(8) will start the specified "getty" command on this entry).

TTY_SECURE
Allows root to login on this terminal. Note: **TTY_ON** must be included for this to be useful.

ty_window is the command to execute for a window system associated with the line. The window system will be started before the command specified in the **ty_getty** entry is executed. If none is specified, this will be NULL.

ty_comment is the trailing comment field, if any; a leading delimiter and white space will be removed.

getttyent( ) reads the next line from the **ttytab** file, opening the file if necessary; **setttyent( )** rewinds the file; **endttyent( )** closes it.

getttynam( ) searches from the beginning of the file until a matching *name* is found (or until EOF is encountered).

FILES

/etc/ttytab

SEE ALSO

login(1), **ttyslot**(3V), **gettytab**(5), **ttytab**(5), **termcap**(5), **getty**(8), **init**(8)

DIAGNOSTICS

NULL pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getusershell, setusershell, endusershell – get legal user shells

SYNOPSIS

char *getusershell( )

setusershell( )

endusershell( )

DESCRIPTION

getusershell( ) returns a pointer to a legal user shell as defined by the system manager in the file **/etc/shells**. If **/etc/shells** does not exist, the four locations of the two standard system shells **/bin/sh**, **/bin/csh**, **/usr/bin/sh** and **/usr/bin/csh** are returned.

getusershell( ) reads the next line (opening the file if necessary); **setusershell( )** rewinds the file; **endusershell( )** closes it.

FILES

/etc/shells
/bin/sh
/bin/csh
/usr/bin/sh
/usr/bin/csh

DIAGNOSTICS

The routine **getusershell( )** returns a NULL pointer (0) on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getwd – get current working directory pathname

SYNOPSIS

#include <sys/param.h>

char *getwd(pathname)
char pathname[MAXPATHLEN];

DESCRIPTION

getwd( ) copies the absolute pathname of the current working directory to *pathname* and returns a pointer to the result.

DIAGNOSTICS

getwd( ) returns zero and places a message in *pathname* if an error occurs.

NAME

hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS

#include <search.h>

ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy ( )

DESCRIPTION

hsearch( ) is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. *item* is a structure of type ENTRY (defined in the **<search.h>** header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) *action* is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

hcreate( ) allocates sufficient space for the table, and must be called before **hsearch( )** is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

hdestroy( ) destroys the search table, and may be followed by another call to **hcreate**.

NOTES

hsearch( ) uses **open addressing** with a *multiplicative* hash function.

EXAMPLE

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```
#include <stdio.h>
#include <search.h>
struct info {           /* this is the info stored in the table */
        int age, room;  /* other than the key. */
};
#define
NUM_EMPL    5000    /* # of elements in search table */
main( )
{
        /* space to store strings */
        char string_space[NUM_EMPL*20];
        /* space to store employee info */
        struct info info_space[NUM_EMPL];
        /* next avail space in string_space */
        char *str_ptr = string_space;
        /* next avail space in info_space */
        struct info *info_ptr = info_space;
        ENTRY item, *found_item, *hsearch( );
        /* name to look for in table */
        char name_to_find[30];
        int i = 0;
        /* create table */
```

```
        (void) hcreate(NUM_EMPL);
        while (scanf("%s%d%d", str_ptr, &info_ptr->age,
             &info_ptr->room) !=
EOF && i++ <
NUM_EMPL) {
                /* put info in structure, and structure in item */
                item.key = str_ptr;
                item.data = (char *)info_ptr;
                str_ptr += strlen(str_ptr) + 1;
                info_ptr++;
                /* put item into table */
                (void) hsearch(item,
ENTER);
        }
        /* access table */
        item.key = name_to_find;
        while (scanf("%s", item.key) != EOF) {
            if ((found_item = hsearch(item,
FIND)) != NULL) {
                /* if item is in the table */
                 (void)printf("found %s, age = %d, room = %d\n",
                        found_item->key,
                        ((struct info *)found_item->data)->age,
                        ((struct info *)found_item->data)->room);
            } else {
                 (void)printf("no such employee %s\n",
                        name_to_find);
            }
        }
}
```

SEE ALSO

bsearch(3), **lsearch**(3), **malloc**(3V), **string**(3), **tsearch**(3)

DIAGNOSTICS

hsearch( ) returns a NULL pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

hcreate( ) returns zero if it cannot allocate sufficient space for the table.

WARNING

hsearch( ) and **hcreate**( ) use **malloc**(3V) to allocate space.

BUGS

Only one hash search table may be active at any given time.

NAME

inet inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long
inet_addr(cp)
char *cp;

inet_network(cp)
char *cp;

struct in_addr
inet_makeaddr(net, lna)
int net, lna;

inet_lnaof(in)
struct in_addr in;

inet_netof(in)
struct in_addr in;

char *
inet_ntoa(in)
struct in_addr in;
```

DESCRIPTION

The routines **inet_addr()** and **inet_network()** each interpret character strings representing numbers expressed in the Internet standard '.' notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine **inet_makeaddr()** takes an Internet network number and a local network address and constructs an Internet address from it. The routines **inet_netof()** and **inet_lnaof()** break apart Internet host addresses, returning the network number and local network address part, respectively.

The routine **inet_ntoa()** returns a pointer to a string in the base 256 notation ''d.d.d.d'' described below.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Values specified using the '.' notation take one of the following forms:

a.b.c.d
a.b.c
a.b
a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note: when an Internet address is viewed as a 32-bit integer quantity on Sun386i systems, the bytes referred to above appear as **d.c.b.a**. That is, Sun386i bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as ''parts'' in a '.' notation may be decimal, octal, or hexadecimal, as specified in the C language (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

gethostent(3N), **getnetent**(3N), **hosts**(5), **networks**(5),

DIAGNOSTICS

The value −1 is returned by **inet_addr**( ) and **inet_network**( ) for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

The return value from **inet_ntoa**( ) points to static information which is overwritten in each call.

NAME

initgroups – initialize supplementary group IDs

SYNOPSIS

initgroups(name, basegid)
char *name;
int basegid;

DESCRIPTION

initgroups( ) reads through the group file and sets up, using the **setgroups** call (see **getgroups**(2V)), the supplementary group IDs for the user specified in *name*. The **basegid** is automatically included in the supplementary group IDs. Typically this value is given as the group number from the password file.

FILES

/etc/group

SEE ALSO

getgroups(2V), **getgrent**(3V)

DIAGNOSTICS

initgroups( ) returns –1 if it was not invoked by the super-user.

BUGS

initgroups( ) uses the routines based on **getgrent**(3V). If the invoking program uses any of these routines, the group structure will be overwritten in the call to **initgroups**.

NAME

insque, remque – insert/remove element from a queue

SYNOPSIS

```
struct qelem {
        struct  qelem *q_forw;
        struct  qelem *q_back;
        char    q_data[];
};

insque(elem, pred)
struct qelem *elem, *pred;

remque(elem)
struct qelem *elem;
```

DESCRIPTION

insque() and **remque**() manipulate queues built from doubly linked lists. Each element in the queue must be in the form of ''struct qelem''. **insque**() inserts *elem* in a queue immediately after *pred*; **remque**() removes an entry *elem* from a queue.

NAME

issecure – indicates whether system is running secure

SYNOPSIS

int issecure()

DESCRIPTION

This function tells whether the system has been configured to run in secure mode. It returns 0 if the system is not running secure, and non-zero if the system is running secure.

NAME

kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process

SYNOPSIS

```
#include <kvm.h>
#include <sys/param.h>
#include <sys/user.h>
#include <sys/proc.h>

struct user *kvm_getu(kd, proc)
kvm_t *kd;
struct proc *proc;

int kvm_getcmd(kd, proc, u, arg, env)
kvm_t *kd;
struct proc *proc;
struct user *u;
char ***arg;
char ***env;
```

DESCRIPTION

kvm_getu( ) reads the u-area of the process specified by *proc* to an area of static storage associated with *kd* and returns a pointer to it. Subsequent calls to **kvm_getu( )** will overwrite this static area.

kd is a pointer to a kernel identifier returned by **kvm_open**(3K). *proc* is a pointer to a copy (in the current process' address space) of a *proc* structure (obtained, for instance, by a prior **kvm_nextproc**(3K) call).

kvm_getcmd( ) constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by *proc*.

kd is a pointer to a kernel identifier returned by **kvm_open**(3K). *u* is a pointer to a copy (in the current process' address space) of a *user* structure (obtained, for instance, by a prior **kvm_getu( )** call). If *arg* is not NULL, then the command line arguments are formed into a null-terminated array of string pointers. The address of the first such pointer is returned in *arg*. If *env* is not NULL, then the environment is formed into a null-terminated array of string pointers. The address of the first of these is returned in *env*.

The pointers returned in *arg* and *env* refer to data allocated by **malloc**(3V) and should be freed (by a call to **free** (see **malloc**(3V)) when no longer needed. Both the string pointers and the strings themselves are deallocated when freed.

Since the environment and command line arguments may have been modified by the user process, there is no guarantee that it will be possible to reconstruct the original command at all. Thus, **kvm_getcmd( )** will make the best attempt possible, returning –1 if the user process data is unrecognizable.

RETURN VALUES

On success, **kvm_getu( )** returns a pointer to a copy of the u-area of the process specified by *proc*. On failure, it returns NULL.

kvm_getcmd( ) returns:

0 on success.

–1 on failure.

SEE ALSO

execve(2V), **kvm_nextproc**(3K), **kvm_open**(3K), **kvm_read**(3K), **malloc**(3V)

NOTES

If **kvm_getcmd()** returns −1, the caller still has the option of using the command line fragment that is stored in the u-area.

NAME

kvm_getproc, kvm_nextproc, kvm_setproc – read system process structures

SYNOPSIS

```
#include <kvm.h>
#include <sys/param.h>
#include <sys/time.h>
#include <sys/proc.h>

struct proc *kvm_getproc(kd, pid)
kvm_t *kd;
int pid;

struct proc *kvm_nextproc(kd)
kvm_t *kd;

int kvm_setproc(kd)
kvm_t *kd;
```

DESCRIPTION

kvm_nextproc() may be used to sequentially read all of the system process structures from the kernel identified by *kd* (see **kvm_open**(3K)). Each call to **kvm_nextproc()** returns a pointer to the static memory area that contains a copy of the next valid process table entry. There is no guarantee that the data will remain valid across calls to **kvm_nextproc()**, **kvm_setproc()**, or **kvm_getproc()**. Therefore, if the process structure must be saved, it should be copied to non-volatile storage.

For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to **kvm_nextproc()**, and since the cache may contain obsolete information, there is no guarantee that *every* process structure returned refers to an active process, nor is it certain that *all* processes will be reported.

kvm_setproc() rewinds the process list, enabling **kvm_nextproc()** to rescan from the beginning of the system process table. **kvm_setproc()** will always flush the process structure cache, allowing an application to re-scan the process table of a running system.

kvm_getproc() locates the **proc** structure of the process specified by *pid* and returns a pointer to it. **kvm_getproc()** does not interact with the process table pointer manipulated by **kvm_nextproc**, however, the restrictions regarding the validity of the data still apply.

RETURN VALUES

On success, **kvm_nextproc()** returns a pointer to a copy of the next valid process table entry. On failure, it returns NULL.

On success, **kvm_getproc()** returns a pointer to the **proc** structure of the process specified by *pid*. On failure, it returns NULL.

kvm_setproc() returns:

0 on success.

−1 on failure.

SEE ALSO

kvm_getu(3K), **kvm_open**(3K), **kvm_read**(3K)

NAME

kvm_nlist – get entries from kernel symbol table

SYNOPSIS

#include <kvm.h>
#include <nlist.h>

int kvm_nlist(kd, nl)
kvm_t *kd;
struct nlist *nl;

DESCRIPTION

kvm_nlist() examines the symbol table from the kernel image identified by *kd* (see **kvm_open**(3K)) and selectively extracts a list of values and puts them in the array of **nlist()** structures pointed to by *nl*. The name list pointed to by **nl()** consists of an array of structures containing names, types and values. The *n_name* field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a null string) in the *n_name* field. For each entry in *nl*, if the named symbol is present in the kernel symbol table, its value and type are placed in the *n_value* and *n_type* fields. If a symbol cannot be located, the corresponding *n_type* field of **nl()** is set to zero.

RETURN VALUES

On success, **kvm_nlist()** returns the number of symbols that were not located in the symbol table. On failure, it returns −1 and sets all of the *n_type* fields in members of the array pointed to by **nl** to zero.

SEE ALSO

kvm_open(3K), **kvm_read**(3K), **nlist**(3V), **a.out**(5)

NAME

kvm_open, kvm_close – specify a kernel to examine

SYNOPSIS

```
#include <kvm.h>
#include <fcntl.h>

kvm_t *kvm_open(namelist, corefile, swapfile, flag, errstr)
char *namelist, *corefile, *swapfile;
int flag;
char *errstr;

int kvm_close(kd)
kvm_t *kd;
```

DESCRIPTION

kvm_open( ) initializes a set of file descriptors to be used in subsequent calls to kernel VM routines. It returns a pointer to a kernel identifier that must be used as the *kd* argument in subsequent kernel VM function calls.

The *namelist* argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in *corefile*. If *namelist* is NULL, the symbol table of the currently running kernel is used to determine offsets in the core image. In this case, it is up to the implementation to select an appropriate way to resolve symbolic references (for instance, using **/vmunix** as a default *namelist* file).

corefile specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see **savecore**(8)) or the special device **/dev/mem**. If *corefile* is NULL, the currently running kernel is accessed (using **/dev/mem** and **/dev/kmem**).

swapfile specifies a file that represents the swap device. If both *corefile* and *swapfile* are NULL, the swap device of the "currently running kernel" is accessed. Otherwise, if *swapfile* is NULL, **kvm_open()** may succeed but subsequent **kvm_getu**(3K) function calls may fail if the desired information is swapped out.

flag is used to specify read or write access for *corefile* and may have one of the following values:

O_RDONLY	open for reading
O_RDWR	open for reading and writing

errstr is used to control error reporting. If it is a NULL pointer, no error messages will be printed. If it is non-NULL, it is assumed to be the address of a string that will be used to prefix error messages generated by **kvm_open**. Errors are printed to **stderr**. A useful value to supply for *errstr* would be **argv**[0]. This has the effect of printing the process name in front of any error messages.

kvm_close( ) closes all file descriptors that were associated with *kd*. These files are also closed on **exit**(2v) and **execve**(2V). **kvm_close**( ) also resets the **proc** pointer associated with **kvm_nextproc**(3K) and flushes any cached kernel data.

RETURN VALUES

kmv_open( ) returns a non-NULL value suitable for use with subsequent kernel VM function calls. On failure, it returns NULL and no files are opened.

kvm_close( ) returns:

0	on success.
−1	on failure.

FILES

/vmunix
/dev/kmem
/dev/mem
/dev/drum

SEE ALSO

execve(2V), **exit**(2v), **kvm_getu**(3K), **kvm_nextproc**(3K), **kvm_nlist**(3K), **kvm_read**(3K), **savecore**(8)

NAME

kvm_read, kvm_write – copy data to or from a kernel image or running system

SYNOPSIS

```
#include <kvm.h>

int kvm_read(kd, addr, buf, nbytes)
kvm_t *kd;
unsigned long addr;
char *buf;
unsigned nbytes;

int kvm_write(kd, addr, buf, nbytes)
kvm_t *kd;
unsigned long addr;
char *buf;
unsigned nbytes;
```

DESCRIPTION

kvm_read() transfers data from the kernel image specified by *kd* (see **kvm_open**(3K)) to the address space of the process. *nbytes* bytes of data are copied from the kernel virtual address given by *addr* to the buffer pointed to by *buf*.

kvm_write() is like **kvm_read()**, except that the direction of data transfer is reversed. In order to use this function, the **kvm_open**(3K) call that returned *kd* must have specified write access. If a user virtual address is given, it is resolved in the address space of the process specified in the most recent **kvm_getu**(3K) call.

RETURN VALUES

On success, **kvm_read()** and **kvm_write()** return the number of bytes actually transferred. On failure, they return –1.

SEE ALSO

kvm_getu(3K), **kvm_nlist**(3K), **kvm_open**(3K)

NAME

l3tol, ltol3 – convert between 3-byte integers and long integers

SYNOPSIS

```
#include <stdlib.h>
void l3tol (lp, cp, n)
long *lp;
const char *cp;
int n;

void ltol3 (cp, lp, n)
char *cp;
const long *lp;
int n;
```

DESCRIPTION

l3tol() converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

ltol3() performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for filesystem maintenance where the block numbers are three bytes long.

SEE ALSO

fs(5)

WARNINGS

Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.

NAME

ldahread – read the archive header of a member of a COFF archive file

SYNOPSIS

#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

If **TYPE(***ldptr***)** is the archive file magic number, **ldahread** reads the archive header of the COFF file currently associated with *ldptr* into the area of memory beginning at *arhead*.

ldahread returns **SUCCESS** or **FAILURE**. **ldahread** will fail if **TYPE(***ldptr***)** does not represent an archive file, or if it cannot read the archive header.

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **intro**(5)

NAME

ldclose, ldaclose – close a COFF file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldclose (ldptr)
LDFILE *ldptr;

int ldaclose (ldptr)
LDFILE *ldptr;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldopen(3X) and **ldclose**() are designed to provide uniform access to both simple COFF object files and COFF object files that are members of archive files. Thus an archive of COFF files can be processed as if it were a series of simple COFF files.

If **TYPE**(*ldptr*) does not represent an archive file, **ldclose**() will close the file and free the memory allocated to the **LDFILE** structure associated with *ldptr*. If **TYPE**(*ldptr*) is the magic number of an archive file, and if there are any more files in the archive, **ldclose**() will reinitialize **OFFSET**(*ldptr*) to the file address of the next archive member and return **FAILURE**. The **LDFILE** structure is prepared for a subsequent **ldopen**(3X). In all other cases, **ldclose**() returns SUCCESS.

ldaclose() closes the file and frees the memory allocated to the **LDFILE** structure associated with *ldptr* regardless of the value of **TYPE**(*ldptr*). **ldaclose**() always returns **SUCCESS**. The function is often used in conjunction with *ldaopen*.

The program must be loaded with the object file access routine library **libld.a**.

intro(5) describes *INCDIR* and *LIBDIR*.

SEE ALSO

fclose(3V), **ldfcn**(3), **ldopen**(3X), **intro**(5)

NAME

ldfcn – common object file access routines

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

These routines are for reading COFF object files and archives containing COFF object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function **ldopen**(3X) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE	*ldptr;
TYPE(ldptr)	The file magic number used to distinguish between archive members and simple object files.
IOPTR(ldptr)	The file pointer returned by *fopen* and used by the standard input/output functions.
OFFSET(ldptr)	The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.
HEADER(ldptr)	The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) Functions that open or close an object file

ldopen(3X) and **ldaopen**( ) (see **ldopen**(3X))
open a common object file
ldclose(3X) and **ldaclose**( ) (see **ldclose**(3X))
close a common object file

(2) Functions that read header or symbol table information

ldahread(3X)
read the archive header of a member of an archive file
ldfhread(3X)
read the file header of a common object file
ldshread(3X) and **ldnshread**( ) (see **ldshread**(3X))
read a section header of a common object file
ldtbread(3X)
read a symbol table entry of a common object file
ldgetname(3X)
retrieve a symbol name from a symbol table entry or from the string table

(3) Functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

> **ldohseek**(3X)
> seek to the optional file header of a common object file
> **ldsseek**(3X) and **ldnsseek**( ) (see **ldsseek**(3X))
> seek to a section of a common object file
> **ldrseek**(3X) and **ldnrseek**( ) (see **ldrseek**(3X))
> seek to the relocation information for a section of a common object file
> **ldlseek**(3X) and **ldnlseek**( ) (see **ldlseek**(3X))
> seek to the line number information for a section of a common object file
> **ldtbseek**(3X)
> seek to the symbol table of a common object file

(4) The unction **ldtbindex**(3X), which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except **ldopen**(3X), **ldgetname**(3X), **ldtbindex**(3X) return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h**. **ldopen**(3X) and **ldaopen**( ) (see **ldopen**(3X)) both return pointers to an **LDFILE** structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h**. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the **LDFILE** structure into a reference to its file descriptor field.

The following macros are provided:

```
GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)
```

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

fseek(3S), **ldahread**(3X), **ldclose**(3X), **ldgetname**(3X), **ldfhread**(3X), **ldlread**(3X), **ldlseek**(3X), **ldohseek**(3X), **ldopen**(3X), **ldrseek**(3X), **ldlseek**(3X), **ldshread**(3X), **ldtbindex**(3X), **ldtbread**(3X), **ldtbseek**(3X), **stdio**(3V), **intro**(5)

WARNING

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function **fseek**(3S). **FSEEK** should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members.

NAME

ldfhread – read the file header of a COFF file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldfhread() reads the file header of the COFF file currently associated with *ldptr* into the area of memory beginning at *filehead*.

ldfhread() returns **SUCCESS** or **FAILURE**. **ldfhread()** will fail if it cannot read the file header.

In most cases the use of **ldfhread()** can be avoided by using the macro **HEADER(***ldptr***)** defined in **ldfcn.h** (see **ldfcn**(3)). The information in any field, *fieldname*, of the file header may be accessed using **HEADER(ldptr).fieldname.**

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X)

NAME

ldgetname – retrieve symbol name for COFF file symbol table entry

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldgetname( ) returns a pointer to the name associated with **symbol** as a string. The string is contained in a static buffer local to **ldgetname( )** that is overwritten by each call to **ldgetname( )**, and therefore must be copied by the caller if the name is to be saved.

ldgetname( ) can be used to retrieve names from object files without any backward compatibility problems. **ldgetname( )** will return NULL (defined in **stdio.h**) for an object file if the name cannot be retrieved. This situation can occur:

- if the ''string table'' cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to **ldgetname( )** that looks like a reference to a name in a nonexistent string table), or
- if the name's offset into the string table is past the end of the string table.

Typically, **ldgetname( )** will be called immediately after a successful call to **ldtbread( )** to retrieve the name associated with the symbol table entry filled by **ldtbread( )**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X), **ldtbseek**(3X)

NAME

ldlread, ldlinit, ldlitem – manipulate line number entries of a COFF file function

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO *linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO *linent;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldlread( ) searches the line number entries of the COFF file currently associated with *ldptr*. **ldlread**( ) begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcnindx*, the index of its entry in the object file symbol table. **ldlread**( ) reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at *linent*.

ldlinit( ) and **ldlitem**( ) together perform exactly the same function as **ldlread**( ). After an initial call to **ldlread**( ) or **ldlinit**( ), **ldlitem**( ) may be used to retrieve a series of line number entries associated with a single function. **ldlinit**( ) simply locates the line number entries for the function identified by *fcnindx*. **ldlitem**( ) finds and reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at **linent**( ).

ldlread( ), **ldlinit**( ), and **ldlitem**( ) each return either SUCCESS or FAILURE. **ldlread**( ) will fail if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*. **ldlinit**( ) will fail if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. **ldlitem**( ) will fail if it finds no line number equal to or greater than *linenum*.

The programs must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldtbindex**(3X)

NAME

ldlseek, ldnlseek – seek to line number entries of a section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldlseek() seeks to the line number entries of the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

ldnlseek() seeks to the line number entries of the section specified by *sectname*.

ldlseek() and **ldnlseek**() return **SUCCESS** or **FAILURE**. **ldlseek**() will fail if *sectindx* is greater than the number of sections in the object file; **ldnlseek**() will fail if there is no section name corresponding with **sectname*. Either function will fail if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of **one**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

NAME

ldohseek – seek to the optional file header of a COFF file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldohseek( ) seeks to the optional file header of the COFF file currently associated with *ldptr*.

ldohsee( ) returns SUCCESS or **FAILURE**. **ldohseek**( ) will fail if the object file has no optional header or if it cannot seek to the optional header.

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldfhread**(3X)

NAME

ldopen, ldaopen – open a COFF file for reading

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldopen( ) and **ldclose**(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of COFF files can be processed as if it were a series of simple COFF files.

If *ldptr* has the value NULL, then **ldopen**( ) will open *filename* and allocate and initialize the **LDFILE** structure, and return a pointer to the structure to the calling program.

If *ldptr* is valid and if **TYPE**(*ldptr*) is the archive magic number, **ldopen**( ) will reinitialize the **LDFILE** structure for the next archive member of *filename*.

ldopen( ) and **ldclose**(3X) are designed to work in concert. *ldclose* will return **FAILURE** only when **TYPE**(*ldptr*) is the archive magic number and there is another file in the archive to be processed. Only then should **ldopen**( ) be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, **ldopen**( ) should be called with a NULL *ldptr* argument.

The following is a prototype for the use of **ldopen**( ) and **ldclose**(3X).

```
/* for each filename to be processed */

ldptr = NULL;
do
{
        if ( (ldptr = ldopen(filename, ldptr)) != NULL )
        {
                /* check magic number */
                /* process the file */
        }
} while (ldclose(ldptr) == FAILURE );
```

If the value of *oldptr* is not NULL, **ldaopen**( ) will open *filename* anew and allocate and initialize a new **LDFILE** structure, copying the **TYPE**, **OFFSET**, and **HEADER** fields from *oldptr*. **ldaopen**( ) returns a pointer to the new **LDFILE** structure. This new pointer is independent of the old pointer, *oldptr*. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both **ldopen( )** and **ldaopen( )** open *filename* for reading. Both functions return NULL if *filename* cannot be opened, or if memory for the **LDFILE** structure cannot be allocated. A successful open does not insure that the given file is a COFF file or an archived object file.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

fopen(3V), **ldclose**(3X), **ldfcn**(3)

NAME

ldrseek, ldnrseek – seek to relocation entries of a section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldrseek() seeks to the relocation entries of the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

ldnrseek() seeks to the relocation entries of the section specified by *sectname*.

ldrseek() and **ldnrseek()** return **SUCCESS** or **FAILURE**. **ldrseek()** will fail if *sectindx* is greater than the number of sections in the object file; **ldnrseek()** will fail if there is no section name corresponding with *sectname*. Either function will fail if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note: the first section has an index of **one**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

NAME

ldshread, ldnshread – read an indexed/named section header of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldshread( ) reads the section header specified by *sectindx* of the COFF file currently associated with *ldptr* into the area of memory beginning at *secthead*.

ldnshread( ) reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

ldshread( ) and **ldnshread**( ) return SUCCESS or **FAILURE**. **ldshread**( ) will fail if *sectindx* is greater than the number of sections in the object file; **ldnshread**( ) will fail if there is no section name corresponding with *sectname*. Either function will fail if it cannot read the specified section header.

Note: the first section header has an index of *one*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X)

NAME

ldsseek, ldnsseek – seek to an indexed/named section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldsseek( ) seeks to the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

ldnsseek( ) seeks to the section specified by *sectname*.

ldsseek( ) and **ldnsseek**( ) return **SUCCESS** or **FAILURE**. **ldsseek**( ) will fail if *sectindx* is greater than the number of sections in the object file; **ldnsseek**( ) will fail if there is no section name corresponding with *sectname*. Either function will fail if there is no section data for the specified section or if it cannot seek to the specified section.

Note: the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

NAME

ldtbindex – compute the index of a symbol table entry of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;
```

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldtbindex( ) returns the (**long**) index of the symbol table entry at the current position of the COFF file associated with *ldptr*.

The index returned by **ldtbindex**( ) may be used in subsequent calls to **ldtbread**(3X). However, since **ldtbindex** ( ) returns the index of the symbol table entry that begins at the current position of the object file, if **ldtbindex**( ) is called immediately after a particular symbol table entry has been read, it will return the index of the next entry.

ldtbindex( ) will fail if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X), **ldtbseek**(3X)

NAME

ldtbread – read an indexed symbol table entry of a COFF file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldtbread() reads the symbol table entry specified by *symindex* of the COFF file currently associated with *ldptr* into the area of memory beginning at **symbol**.

ldtbread() returns **SUCCESS** or **FAILURE**. **ldtbread()** will fail if *symindex* is greater than or equal to the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note: the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldtbseek**(3X), **ldgetname**(3X)

NAME

ldtbseek – seek to the symbol table of a COFF file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ldtbseek( ) seeks to the symbol table of the COFF file currently associated with *ldptr*.

ldtbseek( ) returns **SUCCESS** or **FAILURE**. **ldtbseek( )** will fail if the symbol table has been stripped from the object file, or if it cannot seek to the symbol table.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X)

NAME

localdtconv – get date and time formatting conventions

SYNOPSIS

#include <locale.h>

struct dtconv *localdtconv( )

DESCRIPTION

localdtconv( ) returns a pointer to a structure of type **struct dtconv** containing values appropriate for the formatting of dates and times according to the rules of the current locale.

The members include the following:

char *abbrev_month_names[12]
: The abbreviated names of the months; for example, the abbreviated name for January is **abbrev_month_names[0]** and the abbreviated name for December is **abbrev_month_names[11]**.

char *month_names[12]
: The full names of the months; for example, the full name for January is **month_names[0]** and the full name for December is **month_names[11]**.

char *abbrev_weekday_names[7]
: The abbreviated names of the weekdays; for example, the abbreviated name for Sunday is **abbrev_weekday_names[0]** and the abbreviated name for Saturday is **abbrev_weekday_names[6]**.

char *weekday_names[7]
: The full names of the weekdays; for example, the full name for Sunday is **weekday_names[0]** and the full name for Saturday is **weekday_names[6]**.

char *time_format
: The standard format for times, using the format specifiers supported by **strftime( )** and **strptime( )** (see **ctime**(3V)).

char *sdate_format
: The standard short format for dates, using the format specifiers supported by **ctime (3V).**

char *dtime_format
: The standard short format for dates and times together, using the format specifiers supported by **ctime**(3V).

char *am_string
: The string representing AM.

char *pm_string
: The string representing PM.

char *ldate_format
: The standard long format for dates, using the format specifiers supported by **ctime**(3V).

The values for the members in the C locale are:

abbrev_month_names	**Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec**
month_names	**January, February, March, April, May, June, July, August, September, October, November, December**
abbrev_weekday_names	**Sun, Mon, Tue, Wed, Thu, Fri, Sat**
weekday_names	**Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday**
time_format	**%H:%M:%S**

sdate_format	**%m/%d/%y**
dtime_format	**%a %b %e %T %Z %Y**
am_string	**AM**
pm_string	**PM**
ldate_format	**%A, %B %e, %Y**

FILES

/usr/share/lib/locale/LC_TIME
standard locale information directory for category **LC_TIME**

SEE ALSO

ctime(3V), **setlocale**(3V)

NAME

localeconv – get numeric and monetary formatting conventions

SYNOPSIS

```
#include <limits.h>
#include <locale.h>

struct lconv *localeconv( )
```

DESCRIPTION

localeconv( ) returns a pointer to a structure of type **struct lconv** containing values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type (**char ***) are strings; if a string has the value "", the value is not available in the current locale or has zero length. The members with type **char** are nonnegative numbers; if any of them have the value **CHAR_MAX** the value is not available in the current locale. The **lconv** structure is defined in **<locale.h>** as follows:

```
struct lconv {
        char    *decimal_point;         /* decimal point character */
        char    *thousands_sep;         /* thousands separator character */
        char    *grouping;              /* grouping of digits */
        char    *int_curr_symbol;       /* international currency symbol */
        char    *currency_symbol;       /* local currency symbol */
        char    *mon_decimal_point;     /* monetary decimal point character */
        char    *mon_thousands_sep;     /* monetary thousands separator */
        char    *mon_grouping;          /* monetary grouping of digits */
        char    *positive_sign;         /* monetary credit symbol */
        char    *negative_sign;         /* monetary debit symbol */
        char    int_frac_digits;        /* intl monetary number of fractional digits */
        char    frac_digits;            /* monetary number of fractional digits */
        char    p_cs_precedes;          /* true if currency symbol precedes credit */
        char    p_sep_by_space;         /* true if space separates c.s. from credit */
        char    n_cs_precedes;          /* true if currency symbol precedes debit */
        char    n_sep_by_space;         /* true if space separates c.s. from debit */
        char    p_sign_posn;            /* position of sign for credit */
        char    n_sign_posn;            /* position of sign for debit */
};
```

The fields of this structure represent:

decimal_point
The decimal-point character used to format non-monetary quantities.

thousands_sep
The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.

grouping
A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.

int_curr_symbol
The international currency symbol applicable to the current locale, left-justified within a four-character SPACE-padded field. The character sequences are those specified in: *ISO 4217 Codes for the Representation of Currency and Funds*.

currency_symbol
The local currency symbol applicable to the current locale.

mon_decimal_point
The decimal-point used to format monetary quantities.

mon_thousands_sep
The character used to separate groups of digits to the left of the decimal-point character in formatted monetary quantities.

mon_grouping
A string whose elements indicate the size of each group of digits in formatted monetary quantities.

positive_sign
The string used to indicate a nonnegative-valued formatted monetary quantity.

negative_sign
The string used to indicate a negative-valued formatted monetary quantity.

int_frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

frac_digits
The number of fractional digits (those to the right of the decimal-point) to be displayed in a formatted monetary quantity.

p_cs_precedes
1 if the **currency_symbol** precedes the value for a nonnegative formatted monetary quantity; 0 if the **currency_symbol** succeeds the value for a nonnegative formatted monetary quantity.

p_sep_by_space
1 if the **currency_symbol** is separated by a SPACE from the value for a nonnegative formatted monetary quantity; 0 if the **currency_symbol** is not separated by a SPACE from the value for a nonnegative formatted monetary quantity.

n_cs_precedes
1 if the **currency_symbol** precedes the value for a negative formatted monetary quantity; 0 if the **currency_symbol** succeeds the value for a negative formatted monetary quantity.

n_sep_by_space
1 if the **currency_symbol** is separated by a SPACE from the value for a negative formatted monetary quantity; 0 if the **currency_symbol** is not separated by a SPACE from the value for a negative formatted monetary quantity.

p_sign_posn
A value indicating the positioning of the **positive_sign** for a nonnegative formatted monetary quantity.

n_sign_posn
A value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

The elements of **grouping** and **mon_grouping** are interpreted as follows:

CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.

The values of **p_sign_posn** and **n_sign_posn** are interpreted as follows:

0	Parentheses surround the quantity and **currency_symbol**.
1	The sign string precedes the quantity and **currency_symbol**.

- **2** The sign string succeeds the quantity and **currency_symbol**.
- **3** The sign string immediately precedes the **currency_symbol**.
- **4** The sign string immediately succeds the **currency_symbol**.

The values for the members in the C locale are:

field	*value*
decimal_point	"."
thousands_sep	""
grouping	""
int_curr_symbol	""
currency_symbol	""
mon_decimal_point	""
mon_thousands_sep	""
mon_grouping	""
positive_sign	""
negative_sign	""
int_frac_digits	CHAR_MAX
frac_digits	CHAR_MAX
p_cs_precedes	CHAR_MAX
p_sep_by_space	CHAR_MAX
n_cs_precedes	CHAR_MAX
n_sep_by_space	CHAR_MAX
p_sign_posn	CHAR_MAX
n_sign_posn	CHAR_MAX

RETURN VALUES

localeconv( ) returns a pointer to **struct lconv** (see **NOTES**).

FILES

/usr/share/lib/locale/LC_MONETARY
standard locale information directory for category **LC_MONETARY**

/usr/share/lib/locale/LC_NUMERIC
standard locale information directory for category **LC_NUMERIC**

SEE ALSO

printf(3V), **scanf**(3V), **setlocale**(3V)

NOTES

localeconv( ) does not modify the **struct lconv** to which it returns a pointer, but subsequent calls to **setlocale**(3V) with categories **LC_ALL**, **LC_MONETARY**, or **LC_NUMERIC** may overwrite the contents of the structure.

NAME

lockf – record locking on files

SYNOPSIS

```
#include <unistd.h>

int lockf(fd, cmd, size)
int fd, cmd;
long size;
```

DESCRIPTION

lockf() places, removes, and tests for exclusive locks on sections of files. These locks are either advisory or mandatory depending on the mode bits of the file. The lock is mandatory if the set-GID bit (**S_ISGID**) is set and the group execute bit (**S_IXGRP**) is clear (see **stat**(2V) for information about mode bits). Otherwise, the lock is advisory.

If a process holds a mandatory exclusive lock on a segment of a file, both read and write operations block until the lock is removed (see **WARNINGS**).

An advisory lock does not affect read and write access to the locked segment. Advisory locks may be used by cooperating processes checking for locks using **F_GETLCK** and voluntarily observing the indicated read and write restrictions.

A locking call on an already locked file section fails, returning an error value or putting the call to sleep until that file section is unlocked. All the locks on a process are removed when that process terminates. See **fcntl**(2V) for more information about record locking.

fd is an open file descriptor. It must have **O_WRONLY** or **O_RDWR** permission for a successful locking call.

cmd is a control value which specifies the action to be taken. The accepted values for *cmd* are defined in **<unistd.h>** as follows:

```
#define F_ULOCK    0    /* Unlock a previously locked section */
#define F_LOCK     1    /* Lock a section for exclusive use */
#define F_TLOCK    2    /* Test and lock a section (non-blocking) */
#define F_TEST     3    /* Test section for other process' locks */
```

F_TEST returns −1 and sets **errno** to EACCES if a lock by another process already exists on the specified section. Otherwise, it returns 0. **F_LOCK** and **F_TLOCK** lock available file sections. **F_ULOCK** removes locks from file sections.

All other values of *cmd* are reserved for future applications and, until implemented, return an error.

size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward *size* bytes if *size* is positive, and extends backward *size* bytes (the preceding bytes up to but not including the current offset) if *size* is negative. If *size* is zero, the section from the current offset through the largest file offset is locked (that is, from the current offset through the present or any future EOF). An area need not be allocated to the file to be locked, such a lock may exist after the EOF.

Sections locked with **F_LOCK** or **F_TLOCK** may contain all or part of an already locked section. They may also be partially or completely contained by an already locked section. Where these overlapping or adjacent locked sections occur, they are combined into a single section. If the table of active locks is full, a lock request requiring an additional table entry fails and an error value is returned.

F_LOCK and **F_TLOCK** differ only in their response to requests for unavailable resources. If a section is already locked, **F_LOCK** directs the calling process to sleep until the resource is available, **F_TLOCK** directs the function to return −1 and set **errno** to EACCES (see **ERRORS**).

When a **F_ULOCK** request releases part of a section with overlapping locks, the remaining section or sections retain the lock. If **F_ULOCK** removes the center of a locked section, the two separate locked sections remain, but an additional element is required in the table of active locks. If this table is full, **errno** is set to ENOLCK and the requested section is not released.

The danger of a deadlock exists when a process controlling a locked resource is put to sleep by requesting an unavailable resource. To avoid this danger, **lockf()** and **fcntl()** scan for this conflict before putting a locked resource to sleep. If a deadlock would result, an error value is returned.

The sleep process can be interrupted with any signal. **alarm**(3V) may be used to provide a timeout facility where needed.

RETURN VALUES

lockf() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES *cmd* is **F_TLOCK** or **F_TEST** and the section is already locked by another process.

Note: In future, **lockf()** may generate EAGAIN under these conditions, so applications testing for EACCES should also test for EAGAIN.

EBADF *fd* is not a valid open descriptor.

cmd is **F_LOCK** or **F_TLOCK** and the process does not have write permission on the file.

EDEADLK *cmd* is **F_LOCK** and a deadlock would occur.

EINTR *cmd* is **F_LOCK** and a signal interrupted the process while it was waiting to complete the lock.

ENOLCK *cmd* is **F_LOCK**, **F_TLOCK**, or **F_ULOCK** and there are no more file lock entries available.

SEE ALSO

chmod(2V), **fcntl**(2V), **flock**(2), **fork**(2V), **alarm**(3V), **lockd**(8C)

WARNINGS

Mandatory record locks are dangerous. If a runaway or otherwise out-of-control process should hold a mandatory lock on a file critical to the system and fail to release that lock, the entire system could hang or crash. For this reason, mandatory record locks may be removed in a future SunOS release.n Use advisory record locking whenever possible.

NOTES

A child process does not inherit locks from its parent on **fork**(2V).

BUGS

lockf() locks do not interact in any way with locks granted by **flock()**, but are compatible with locks granted by **fcntl()**.

NAME

lsearch, lfind – linear search and update

SYNOPSIS

```
#include <stdio.h>
#include <search.h>

char *lsearch (key, base, nelp, width, compar)
char *key;
char *base;
unsigned int *nelp;
unsigned int width;
int (*compar)( );

char *lfind (key, base, nelp, width, compar)
char *key;
char *base;
unsigned int *nelp;
unsigned int width;
int (*compar)( );
```

DESCRIPTION

lsearch() is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. *key* points to the datum to be sought in the table. *base* points to the first element in the table. *nelp* points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. *compar* is the name of the comparison function which the user must supply (**strcmp()**, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

lfind() is the same as **lsearch()** except that if the datum is not found, it is not added to the table. Instead, a NULL pointer is returned.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

This fragment will read in ≤ TABSIZE strings of length ≤ ELSIZE and store them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>
#define
TABSIZE 50
#define
ELSIZE 120
        char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
        unsigned nel = 0;
        int strcmp( );
        . . .
        while (fgets(line,
        ELSIZE, stdin) != NULL &&
```

```
nel < TABSIZE)
        (void) lsearch(line, (char *)tab, &nel, ELSIZE, strcmp);
. . .
```

SEE ALSO

bsearch(3), **hsearch**(3), **tsearch**(3)

DIAGNOSTICS

If the searched for datum is found, both **lsearch**( ) and **lfind**( ) return a pointer to it. Otherwise, **lfind**( ) returns NULL and **lsearch**( ) returns a pointer to the newly added element.

BUGS

Undefined results can occur if there is not enough room in the table to add a new item.

NAME

madvise – provide advice to VM system

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mman.h>

int madvise(addr, len, advice)
caddr_t addr;
size_t len;
int advice;
```

DESCRIPTION

madvise( ) advises the kernel that a region of user mapped memory in the range [*addr*, *addr* + *len*) will be accessed following a type of pattern. The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified mapping range.

Values for *advice* are defined in **<sys/mman.h>** as:

```
#define MADV_NORMAL     0x0      /* No further special treatment */
#define MADV_RANDOM     0x1      /* Expect random page references */
#define MADV_SEQUENTIAL          0x2/* Expect sequential page references */
#define MADV_WILLNEED   0x3      /* Will need these pages */
#define MADV_DONTNEED   0x4      /* Don't need these pages */
```

MADV_NORMAL
The default system characteristic where accessing memory within the address range causes the system to read data from the mapped file. The kernel reads all data from files into pages which are retained for a period of time as a "cache". System pages can be a scarce resource, so the kernel steals pages from other mappings when needed. This is a likely occurrence but only adversely affects system performance if a large amount of memory is accessed.

MADV_RANDOM
Tells the kernel to read in a minimum amount of data from a mapped file when doing any single particular access. Normally when an address of a mapped file is accessed, the system tries to read in as much data from the file as reasonable, in anticipation of other accesses within a certain locality.

MADV_SEQUENTIAL
Tells the system that addresses in this range are likely to only be accessed once, so the system will free the resources used to map the address range as quickly as possible. This is used in the **cat**(1V) and **cp**(1) utilities.

MADV_WILLNEED
Tells the system that a certain address range is definitely needed, so the kernel will read the specified range into memory immediately. This might be beneficial to programs who want to minimize the time it takes to access memory the first time since the kernel would need to read in from the file.

MADV_DONTNEED
Tells the kernel that the specified address range is no longer needed, so the system immediately frees the resources associated with the address range.

madvise( ) should be used by programs that have specific knowledge of their access patterns over a memory object (for example, a mapped file) and wish to increase system performance.

RETURN VALUES

madvise( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL — *addr* is not a multiple of the page size as returned by **getpagesize**(2).

The length of the specified address range is less than or equal to 0.

advice was invalid.

EIO — An I/O error occurred while reading from or writing to the file system.

ENOMEM — Addresses in the range [*addr*, *addr* + *len*) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.

SEE ALSO

mctl(2), **mmap**(2)

NAME

malloc, free, realloc, calloc, cfree, memalign, valloc, mallocmap, mallopt, mallinfo, malloc_debug, malloc_verify, alloca – memory allocator

SYNOPSIS

```
#include <malloc.h>

char *malloc(size)
unsigned size;

int free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

int cfree(ptr)
char *ptr;

char *memalign(alignment, size)
unsigned alignment;
unsigned size;

char *valloc(size)
unsigned size;

void mallocmap( )

int mallopt(cmd, value)
int cmd, value;

struct mallinfo mallinfo( )

#include <alloca.h>

char *alloca(size)
int size;
```

SYSTEM V SYNOPSIS

```
#include <malloc.h>

void *malloc(size)
size_t size;

void free(ptr)
void *ptr;

void *realloc(ptr, size)
void *ptr;
size_t size;

void *calloc(nelem, elsize)
size_t nelem;
size_t elsize;

void *memalign(alignment, size)
size_t alignment;
size_t size;

void *valloc(size)
size_t size;
```

The XPG2 versions of the functions listed in this section are declared as they are in **SYNOPSIS** above, except **free( )**, which is declared as:

void free(ptr)
char *ptr;

DESCRIPTION

These routines provide a general-purpose memory allocation package. They maintain a table of free blocks for efficient allocation and coalescing of free storage. When there is no suitable space already free, the allocation routines call **sbrk( )** (see **brk**(2)) to get more memory from the system.

Each of the allocation routines returns a pointer to space suitably aligned for storage of any type of object. Each returns a NULL pointer if the request cannot be completed (see **DIAGNOSTICS**).

malloc( ) returns a pointer to a block of at least *size* bytes, which is appropriately aligned.

free( ) releases a previously allocated block. Its argument is a pointer to a block previously allocated by **malloc( )**, **calloc( )**, **realloc( )**, **malloc( )**, or **memalign( )**.

realloc( ) changes the size of the block referenced by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If unable to honor a reallocation request, **realloc( )** leaves its first argument unaltered. For backwards compatibility, **realloc( )** accepts a pointer to a block freed since the most recent call to **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, or **memalign( )**. Note: using **realloc( )** with a block freed *before* the most recent call to **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, or **memalign( )** is an error.

calloc( ) uses **malloc( )** to allocate space for an array of *nelem* elements of size *elsize*, initializes the space to zeros, and returns a pointer to the initialized block. The block can be freed with **free( )** or **cfree( )**.

memalign( ) allocates *size* bytes on a specified alignment boundary, and returns a pointer to the allocated block. The value of the returned address is guaranteed to be an even multiple of *alignment*. Note: the value of *alignment* must be a power of two, and must be greater than or equal to the size of a word.

valloc(*size***)** is equivalent to **memalign(getpagesize( ),** *size***)**.

mallocmap( ) prints a map of the heap to the standard output. **mallocmap( )** prints each block's address, size (in bytes) and status (free or busy). A block must have a size that is no larger than the current extent of the heap.

mallopt( ) allows quick allocation of small blocks of memory. **mallopt( )** tells subsequent calls to **malloc( )** to allocate *holding blocks* containing small blocks. Under this small block algorithm, a request to **malloc( )** for a small block of memory returns a pointer to one of the pre-allocated small blocks. Different holding blocks are created as needed for different sizes of small blocks.

cmd may be one of the following values, defined in **<malloc.h>**:

M_MXFAST Set the maximum size of blocks to be allocated using the small block algorithm (*maxfast*) to *value*. The algorithm allocates all blocks smaller than *maxfast* in large groups and then doles them out very quickly. Initially, *maxfast* is 0 and the small block algorithm is disabled.

M_NLBLKS Set the number of small blocks in a holding block (*numlblks*) to *value*. The holding blocks each contain *numlblks* blocks. *numlblks* must be greater than 1. The default value for *numlblks* is 100.

M_GRAIN Set the granularity for small block requests (*grain*) to *value*. The sizes of all blocks smaller than *maxfast* are rounded up to the nearest multiple of *grain*. *grain* must be greater than 0. The default value of *grain* is the smallest number of bytes which will allow alignment of any data type. When *grain* is set, *value* is rounded up to a multiple of this default.

M_KEEP Preserve data in a freed block until the next **malloc()**, **realloc()**, or **calloc()**. This option is provided only for compatibility with the old version of **malloc()** and is not recommended.

mallopt() may be called repeatedly, but may not be called after the first small block is allocated.

mallinfo() can be used during program development to determine the best settings for the parameters set by **mallopt()**. Do not call **mallinfo()** until after a call to **malloc()**. **mallinfo()** provides information describing space usage. It returns a **mallinfo** structure, defined in **<malloc.h>** as:

```
struct mallinfo {
        int arena;          /* total space in arena */
        int ordblks;        /* number of ordinary blocks */
        int smblks;         /* number of small blocks */
        int hblks;          /* number of holding blocks */
        int hblkhd;         /* space in holding block headers */
        int usmblks;        /* space in small blocks in use */
        int fsmblks;        /* space in free small blocks */
        int uordblks;       /* space in ordinary blocks in use */
        int fordblks;       /* space in free ordinary blocks */
        int keepcost;       /* cost of enabling keep option */

        int mxfast;         /* max size of small blocks */
        int nlblks;         /* number of small blocks in a holding block */
        int grain;          /* small block rounding factor */
        int uordbytes;      /* space (including overhead) allocated in ord. blks */
        int allocated;      /* number of ordinary blocks allocated */
        int treeoverhead;           /* bytes used in maintaining the free tree */
};
```

alloca() allocates *size* bytes of space in the stack frame of the caller, and returns a pointer to the allocated block. This temporary space is automatically freed when the caller returns. Note that if the allocated block is beyond the current stack limit, the resulting behavior is undefined.

malloc(), **realloc()**, **memalign()** and **valloc()** return a non-NULL pointer if *size* is 0, and **calloc()** returns a non-NULL pointer if *nelem* or *elsize* is 0, but these pointers should *not* be dereferenced.

Note: Always cast the value returned by **malloc()**, **realloc()**, **calloc()**, **memalign()**, **valloc()** or **alloca()**.

SYSTEM V DESCRIPTION

The XPG2 versions of **malloc()**, **realloc()**, **memalign()** and **valloc()** return NULL if *size* is 0. The XPG2 version of **calloc()** returns NULL if *nelem* or *elsize* is 0.

RETURN VALUES

On success, **malloc()**, **calloc()**, **realloc()**, **memalign()**, **valloc()** and **alloca()** return a pointer to space suitably aligned for storage of any type of object. On failure, they return NULL.

free() and **cfree()** return:

1 on success.

0 on failure and set **errno** to indicate the error.

mallopt() returns 0 on success. If **mallopt()** is called after the allocation of a small block, or if *cmd* or *value* is invalid, it returns a non-zero value.

mallinfo() returns a **struct mallinfo**.

SYSTEM V RETURN VALUES

If *size* is 0, the XPG2 versions of **malloc( )**, **realloc( )**, **memalign( )** and **valloc( )** return NULL.

If *nelem* or *elsize* is 0, the XPG2 version of **calloc( )** returns NULL.

free( ) does not return a value.

ERRORS

malloc( ), **calloc( )**, **realloc( )**, **valloc( )**, **memalign( )**, **cfree( )**, and **free( )** will each fail if one or more of the following are true:

EINVAL — An invalid argument was specified.

The value of *ptr* passed to **free( )**, **cfree( )**, or **realloc( )** was not a pointer to a block previously allocated by **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, or **memalign( )**.

The allocation heap is found to have been corrupted. More detailed information may be obtained by enabling range checks using **malloc_debug( )**.

ENOMEM — *size* bytes of memory could not be allocated.

FILES

/usr/lib/debug/malloc.o — diagnostic versions of **malloc( )** routines.
/usr/lib/debug/mallocmap.o — routines to print a map of the heap.

SEE ALSO

csh(1), **ld**(1), **brk**(2), **getrlimit**(2), **sigvec**(2), **sigstack**(2)

Stephenson, C.J., *Fast Fits*, in *Proceedings of the ACM 9th Symposium on Operating Systems*, SIGOPS *Operating Systems Review*, vol. 17, no. 5, October 1983.
Core Wars, in *Scientific American*, May 1984.

DIAGNOSTICS

More detailed diagnostics can be made available to programs using **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, **memalign( )**, **cfree( )**, and **free( )**, by including a special relocatable object file at link time (see FILES). This file also provides routines for control of error handling and diagnosis, as defined below. Note: these routines are *not* defined in the standard library.

```
int malloc_debug(level)
int level;

int malloc_verify( )
```

malloc_debug( ) sets the level of error diagnosis and reporting during subsequent calls to **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, **memalign( )**, **cfree( )**, and **free( )**. The value of *level* is interpreted as follows:

Level 0 — **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, **memalign( )**, **cfree( )**, and **free( )** behave the same as in the standard library.

Level 1 — The routines abort with a message to the standard error if errors are detected in arguments or in the heap. If a bad block is encountered, its address and size are included in the message.

Level 2 — Same as level 1, except that the entire heap is examined on every call to the above routines.

malloc_debug( ) returns the previous error diagnostic level. The default level is 1.

malloc_verify( ) attempts to determine if the heap has been corrupted. It scans all blocks in the heap (both free and allocated) looking for strange addresses or absurd sizes, and also checks for inconsistencies in the free space table. **malloc_verify( )** returns 1 if all checks pass without error, and otherwise returns 0. The checks can take a significant amount of time, so it should not be used indiscriminately.

WARNINGS

alloca( ) is machine-, compiler-, and most of all, system-dependent. Its use is strongly discouraged. See **getrlimit**(2), **sigvec**(2), **sigstack**(2), **csh**(1), and **ld**(1).

NOTES

Because **malloc( )**, **realloc( )**, **memalign( )** and **valloc( )** return a non-NULL pointer if *size* is 0, and **calloc( )** returns a non-NULL pointer if *nelem* or *elsize* is 0, a zero size need not be treated as a special case if it should be passed to these functions unpredictably. Also, the pointer returned by these functions may be passed to subsequent invocations of **realloc( )**.

SYSTEM V NOTES

The XPG2 versions of the allocation routines return NULL when passed a zero size (see **SYSTEM V DESCRIPTION** above).

BUGS

Since **realloc( )** accepts a pointer to a block freed since the last call to **malloc( )**, **calloc( )**, **realloc( )**, **valloc( )**, or **memalign( )**, a degradation of performance results. The semantics of **free( )** should be changed so that the contents of a previously freed block are undefined.

NAME

mblen, mbstowcs, mbtowc, wcstombs, wctomb – multibyte character handling

SYNOPSIS

#include <stdlib.h>

```
int mblen(s, n)
char *s;
size_t n;

size_t mbstowcs(s, pwcs, n)
char *s;
wchar_t *pwcs;
size_t n;

int mbtowc(pwc, s, n)
wchar_t *pwc;
char *s;
size_t n;

int wcstombs(s, pwcs, n)
char *s;
wchar_t *pwcs;
size_t n;

int wctomb(s, wchar)
char *s;
wchar_t wcar;
```

DESCRIPTION

The behavior of these functions is affected by the **LC_CTYPE** category of the program's locale. For a stat-dependent encoding, each function is placed into its initial state by a call for which its character pointer argument, *s*, is a NULL pointer. Subsequent calls with *s* as other than a NULL pointer cause the internal stste of the function to be altered as necessary. A call with a *s* as a NULL pointer causes these functions to return a nonzero value if encodings have state dependency, and zero otherwise. After the LC_CTYPE category is changed, the shift state of these functions is indeterminate.

If *s* is not a NULL pointer, these functions work as follows:

mblen()

Determines the number of bytes comprising the multibyte character pointed to by *s*.

mbstowcs()

Converts a sequence of multibyte characters that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding codes and stores no more than *n* codes into the array pointed to by *pwcs*. No multibyte characters that follow a null character (which is converted into a code with value zero) will be examined or converted. Each multibyte character is converted as if by a call to **mbtowc()**, except that the shift state of **mbtowc()** is not affected.

No more than *n* elements will be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

mbtowc()

Determines the number of bytes that comprise the multibyte character pointed to by *s*. **mbtowc()** then determines the code for value of type **wchar_t** that corresponds to that multibyte character. The value of the code corresponding to the null caharacter is zero. If the multibyte character is valid and *pwc* is not a null pointer, **mbtowc()** stores the code in the object pointed to by *pwc*. At most *n* bytes of the array pointed to by *s* will be examined.

wcstowcs( )

Converts a sequence of codes that correspond to multibyte characters from the array pointed to by *pwcs* into a sequence of multibyte characters that begins in the initial shift state and stores these multibyte characters into the array pointed to by *s*, stopping if a multibyte character would exceed the limit of *n* total bytes or if a null character is stored. Each code is converted as if by a call to **wctomb( )**, except that the shift state of **wctomb( )** is not affected.

wctomb( )

Determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is *wchar* (including any change in shift state). **wctomb( )** stores the multibyte character representation in the array object pointed to by *s* (if *s* is not a null pointer). At most, **MB_CUR_MAX** characters are stored. If the value of *wchar* is zero, **wctomb( )** is left in the initial shift state.

RETURN VALUES

If *s* is a null pointer, **mblen( )**, **mbtowc( )**, and **wctomb( )** return a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state dependent encodings.

If *s* is not a null pointer, **mblen( )** and **mbtowc( )** either return 0 (if *s* points to the null character), or return the number of bytes that comprise the converted multibyte character (if the next *n* or fewer bytes form a valid multibyte character), or return −1 (if they do not form a valid multibyte character).

In no case will the value returned by **mbtowc( )** be greater than *n* or the value of the **MB_CUR_MAX** macro. If *s* is not a null pointer, **wctomb( )** returns −1 (if the value does not correspond to a valid multibyte character), or returns the number of bytes that comprise the multibyte character corresponding to *wchar*.

If an invalid multibyte character is encountered, **mbstowcs( )** and **wcstombs( )** return **(size_t)** −1. Otherwise, they return the number of bytes modified, not including a terminating null character, if any.

NAME

memory, memccpy, memchr, memcmp, memcpy, memset – memory operations

SYNOPSIS

```
#include <memory.h>

char *memccpy(s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr(s, c, n)
char *s;
int c, n;

int memcmp(s1, s2, n)
char *s1, *s2;
int n;

char *memcpy(s1, s2, n)
char *s1, *s2;
int n;

char *memset(s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy( ) copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

memchr( ) returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or a NULL pointer if *c* does not occur.

memcmp( ) compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

memcpy( ) copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

memset( ) sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

NOTES

For user convenience, all these functions are declared in the **<memory.h>** header file.

BUGS

memcmp( ) uses native character comparison, which is signed on some machines and unsigned on other machines. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

mktemp, mkstemp – make a unique file name

SYNOPSIS

char *mktemp(template)
char *template;

mkstemp(template)
char *template;

DESCRIPTION

mktemp() creates a unique file name, typically in a temporary filesystem, by replacing *template* with a unique file name, and returns the address of *template*. The string in *template* should contain a file name with six trailing **X**s; **mktemp()** replaces the **X**s with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file. **mkstemp()** makes the same replacement to the template but returns a file descriptor for the template file open for reading and writing. **mkstemp()** avoids the race between testing whether the file exists and opening it for use.

Notes:

- **mktemp()** and **mkstemp()** actually *change* the template string which you pass; this means that you cannot use the same template string more than once — you need a fresh template for every unique file you want to open.
- When **mktemp()** or **mkstemp()** are creating a new unique filename they check for the prior existence of a file with that name. This means that if you are creating more than one unique filename, it is bad practice to use the same root template for multiple invocations of **mktemp()** or **mkstemp()**.

SEE ALSO

getpid(2V), **open**(2V), **tmpfile**(3S), **tmpnam**(3S)

DIAGNOSTICS

mkstemp() returns an open file descriptor upon success. It returns –1 if no suitable file could be created.

mktemp() assigns the null string to *template* when it cannot create a unique name.

BUGS

It is possible to run out of letters.

NAME

mlock, munlock – lock (or unlock) pages in memory

SYNOPSIS

#include <sys/types.h>
int mlock(addr, len) caddr_t addr; size_t len;

int munlock(addr, len)
caddr_t addr;
size_t len;

DESCRIPTION

mlock() uses the mappings established for the address range [*addr, addr* + *len*) to identify memory object pages to be locked in memory. If the page identified by a mapping changes, such as occurs when a copy of a writable **MAP_PRIVATE** page is made upon the first store, the lock will be transferred to the newly copied private page.

munlock() removes locks established with **mlock()**.

A given page may be locked multiple times by executing an **mlock()** through different mappings. That is, if two different processes lock the same page then the page will remain locked until both processes remove their locks. However, within a given mapping, page locks do not nest – multiple **mlock()** operations on the same address in the same process will all be removed with a single **munlock()**. Of course, a page locked in one process and mapped in another (or visible through a different mapping in the locking process) is still locked in memory. This fact can be used to create applications that do nothing other than lock important data in memory, thereby avoiding page I/O faults on references from other processes in the system.

If the mapping through which an **mlock()** has been performed is removed, an **munlock()** is implicitly performed. An **munlock()** is also performed implicitly when a page is deleted through file removal or truncation.

Locks established with **mlock()** are not inherited by a child process after a **fork**(2V).

Due to the impact on system resources, the use of **mlock()** and **munlock()** is restricted to the super-user. Attempts to **mlock()** more memory than a system-specific limit will fail.

RETURN VALUES

mlock() and **munlock()** return:

0 on success.

–1 on failure and set **errno** to indicate the error.

ERRORS

EAGAIN (**mlock()** only.) Some or all of the memory identified by the range [*addr, addr* + *len*) could not be locked due to insufficient system resources.

EINVAL *addr* is not a multiple of the page size as returned by **getpagesize**(2).

ENOMEM Addresses in the range [*addr, addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

EPERM The process's effective user ID is not super-user.

SEE ALSO

fork(2V), **mctl**(2), **mlockall**(3), **mmap**(2), **munmap**(2)

NAME

mlockall, munlockall – lock (or unlock) address space

SYNOPSIS

#include <sys/mman.h>

int mlockall(flags)
int flags;

int munlockall( )

DESCRIPTION

mlockall() locks all pages mapped by an address space in memory. The value of *flags* determines whether the pages to be locked are simply those currently mapped by the address space, those that will be mapped in the future, or both. *flags* is built from the options defined in **<sys/mman.h>** as:

```
#define MCL_CURRENT    0x1     /* lock current mappings */
#define MCL_FUTURE     0x2     /* lock future mappings */
```

If **MCL_FUTURE** is specified to **mlockall()** , then as mappings are added to the address space (or existing mappings are replaced) they will also be locked, provided sufficient memory is available.

Mappings locked via **mlockall()** with any option may be explicitly unlocked with a **munlock( )** call.

munlockall( ) removes address space locks and locks on mappings in the address space.

All conditions and constraints on the use of locked memory as exist for **mlock( )** apply to **mlockall() .**

RETURN VALUES

mlockall( ) and **munlockall( )** return:

0 on success.

–1 on failure and set **errno** to indicate the error.

ERRORS

EAGAIN	(**mlockall( )** only.) Some or all of the memory in the address space could not be locked due to sufficient resources.
EINVAL	*flags* contains values other than MCL_CURRENT and MCL_FUTURE.
EPERM	The process's effective user ID is not super-user.

SEE ALSO

mctl(2), **mlock**(3), **mmap**(2)

NAME

monitor, monstartup, moncontrol – prepare execution profile

SYNOPSIS

```
#include <a.out.h>

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[ ];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)
```

DESCRIPTION

There are two different forms of monitoring available. An executable program created by '**cc –p**' automatically includes calls for the **prof**(1) monitor, and includes an initial call with default parameters to its start-up routine **monstartup**. In this case, **monitor**( ) need not be called explicitly, except to gain fine control over **profil**(2) buffer allocation. An executable program created by '**cc –pg**' automatically includes calls for the **gprof**(1) monitor.

monstartup( ) is a high-level interface to **profil**(2). *lowpc* and *highpc* specify the address range that is to be sampled; the lowest address sampled is that of *lowpc* and the highest is just below *highpc*. **monstartup**( ) allocates space using **sbrk** (see **brk**(2)) and passes it to **monitor**( ) (as described below) to record a histogram of program-counter values, and calls to certain functions. Only calls to functions compiled with '**cc –p**' are recorded.

On Sun-2, Sun-3, and Sun-4 systems, an entire program can be profiled with:

```
extern etext( );
...
monstartup(N_TXTOFF(0), etext);
```

On Sun386i systems, the equivalent code sequence is:

```
extern etext( );
extern _start( );
...
monstartup(_start, etext);
```

etext lies just above all the program text, see **end**(3).

To stop execution monitoring and post results to the file **mon.out**, use:

```
monitor(0);
```

prof(1) can then be used to examine the results.

moncontrol( ) is used to selectively control profiling within a program. This works with both **prof**(1) and **gprof**(1). Profiling begins when the program starts. To stop the collection of profiling statistics, use:

```
moncontrol(0)
```

To resume the collection of statistics, use:

```
moncontrol(1)
```

This allows you to measure the cost of particular functions. Note: an output file is be produced upon program exit, regardless of the state of **moncontrol**.

monitor( ) is a low level interface to **profil**(2). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. At most *nfunc* call counts can be kept.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. **monitor()** divides the buffer into space to record the histogram of program counter samples over the range *lowpc* to *highpc*, and space to record call counts of functions compiled with the **cc −p**.

To profile the entire program on Sun-2, Sun-3, and Sun-4 systems using the low-level interface to **profil**(2), it is sufficient to use

```
extern etext( );
...
monitor(N_TXTOFF(0), etext, buf, bufsize, nfunc);
```

On Sun386i systems, the equivalent calls are:

```
extern etext( );
extern _start( );
...
monitor(_start, etext, buf, bufsize, nfunc);
```

FILES

mon.out

SEE ALSO

cc(1V), **prof**(1), **gprof**(1), **brk**(2), **profil**(2), **end**(3)

NAME

mp, madd, msub, mult, mdiv, mcmp, min, mout, pow, gcd, rpow, itom, xtom, mtox, mfree – multiple precision integer arithmetic

SYNOPSIS

```
#include <mp.h>

madd(a, b, c)
MINT *a, *b, *c;

msub(a, b, c)
MINT *a, *b, *c;

mult(a, b, c)
MINT *a, *b, *c;

mdiv(a, b, q, r)
MINT *a, *b, *q, *r;

mcmp(a,b)
MINT *a, *b;

min(a)
MINT *a;

mout(a)
MINT *a;

pow(a, b, c, d)
MINT *a, *b, *c, *d;

gcd(a, b, c)
MINT *a, *b, *c;

rpow(a, n, b)
MINT *a, *b;
short n;

msqrt(a, b, r)
MINT *a, *b, *r;

sdiv(a, n, q, r)
MINT *a, *q;
short n, *r;

MINT *itom(n)
short n;

MINT *xtom(s)
char *s;

char *mtox(a)
MINT *a;

void mfree(a)
MINT *a;
```

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type MINT. Pointers to a MINT should be initialized using the function **itom()**, which sets the initial value to *n*. Alternatively, **xtom()** may be used to initialize a MINT from a string of hexadecimal digits. **mfree()** may be used to release the storage allocated by the **itom()** and **xtom()** routines.

madd(), **msub()** and **mult()** assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. **mdiv()** assigns the quotient and remainder, respectively, to its third and fourth arguments. **sdiv()** is like **mdiv()** except that the divisor is an ordinary integer. **msqrt** produces the square root and remainder of its first argument. **mcmp()** compares the values of its arguments and returns 0 if the two values are equal, a value greater than 0 if the first argument is greater than the second, and a value less than 0 if the second argument is greater than the first. **rpow** raises *a* to the *n*th power and assigns this value to *b*. **pow()** raises *a* to the *b*th power, reduces the result modulo *c* and assigns this value to *d*. **min()** and **mout()** do decimal input and output. **gcd()** finds the greatest common divisor of the first two arguments, returning it in the third argument. **mtox()** provides the inverse of **xtom()**. To release the storage allocated by **mtox()**, use **free()** (see **malloc**(3V)).

Use the **–lmp** loader option to obtain access to these functions.

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

FILES

/usr/lib/libmp.a

SEE ALSO

malloc(3V)

NAME

msync – synchronize memory with physical storage

SYNOPSIS

#include <sys/types.h>
#include <sys/mman.h>

int msync(addr, len, flags)
caddr_t addr; size_t len; int flags;

DESCRIPTION

msync() writes all modified copies of pages over the range [*addr*, *addr* + *len*) to their permanent storage locations. **msync()** optionally invalidates any copies so that further references to the pages will be obtained by the system from their permanent storage locations.

Values for *flags* are defined in **<sys/mman.h>** as:

```
#define MS_ASYNC        0x1    /* Return immediately */
#define MS_INVALIDATE   0x2    /* Invalidate mappings */
```

and are used to control the behavior of **msync()**. One or more flags may be specified in a single call.

MS_ASYNC returns immediately once all I/O operations are scheduled; normally, **msync()** will not return until all I/O operations are complete. **MS_INVALIDATE** invalidates all cached copies of data from memory objects, requiring them to be re-obtained from the object's permanent storage location upon the next reference.

msync() should be used by programs that require a memory object to be in a known state, for example in building transaction facilities.

RETURN VALUES

msync() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL	*addr* is not a multiple of the page size as returned by **getpagesize**(2).
	flags is not some combination of **MS_ASYNC** or **MS_INVALIDATE.**
EIO	An I/O error occurred while reading from or writing to the file system.
ENOMEM	Addresses in the range [*addr*, *addr* + *len*) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.
EPERM	**MS_INVALIDATE** was specified and one or more of the pages is locked in memory.

SEE ALSO

mctl(2), **mmap**(2)

NAME

ndbm, dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr – data base subroutines

SYNOPSIS

```
#include <ndbm.h>

typedef struct {
char *dptr;
int dsize;
} datum;

DBM *dbm_open(file, flags, mode)
char *file;
int flags, mode;

void dbm_close (db)
DBM *db;

datum dbm_fetch(db, key)
DBM *db;
datum key;

int dbm_store(db, key, content, flags)
DBM *db;
datum key, content;
int flags;

int dbm_delete(db, key)
DBM *db;
datum key;

datum dbm_firstkey(db)
DBM *db;

datum dbm_nextkey(db)
DBM *db;

int dbm_error(db)
DBM *db;

int dbm_clearerr(db)
DBM *db;
```

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. This package replaces the earlier **dbm**(3X) library, which managed only a single database.

keys and *contents* are described by the **datum** typedef. A **datum** specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by **dbm_open**. This will open and/or create the files *file*.**dir** and *file*.**pag** depending on the flags parameter (see **open**(2V)).

A database is closed by calling **dbm_close**.

Once open, the data stored under a key is accessed by **dbm_fetch()** and data is placed under a key by **dbm_store**. The *flags* field can be either **DBM_INSERT** or **DBM_REPLACE**. **DBM_INSERT** will only insert new entries into the database and will not change an existing entry with the same key. **DBM_REPLACE** will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by **dbm_delete**. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **dbm_firstkey()** and **dbm_nextkey**. **dbm_firstkey()** will return the first key in the database. **dbm_nextkey()** will return the next key in the database. This code will traverse the data base:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

dbm_error() returns non-zero when an error has occurred reading or writing the database. **dbm_clearerr()** resets the error condition on the named database.

SEE ALSO

ar(1V), **cat**(1V), **cp**(1), **tar**(1), **open**(2V), **dbm**(3X)

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero return indicates no error. Routines that return a **datum** indicate errors with a NULL **(0)** *dptr*. If **dbm_store** called with a *flags* value of **DBM_INSERT** finds an existing entry with the same key it returns 1.

BUGS

The **.pag** file will contain holes so that its apparent size is about four times its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (**cp**(1), **cat**(1V), **tar**(1), **ar**(1V)) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes). Moreover all key/content pairs that hash together must fit on a single block. **dbm_store()** will return an error in the event that a disk block fills with inseparable data.

dbm_delete() does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **dbm_firstkey()** and **dbm_nextkey()** depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

NAME

nice – change nice value of a process

SYNOPSIS

int nice(incr)

DESCRIPTION

The nice value of the process is changed by *incr*. Positive nice values get less service than normal. See **nice**(1) for a discussion of the relationship of nice value and scheduling priority.

A nice value of 10 is recommended to users who wish to execute long-running programs without undue impact on system performance.

Negative increments are illegal, except when specified by the super-user. The nice value is limited to the range –20 (most urgent) to 19 (least). Requests for values above or below these limits result in the nice value being set to the corresponding limit.

The nice value of a process is passed to a child process by **fork**(2V). For a privileged process to return to normal nice value from an unknown state, **nice**() should be called successively with arguments –40 (goes to nice value –20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SYSTEM V DESCRIPTION

The maximum allowed value for *incr* is 40 (least urgent).

RETURN VALUES

nice() returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

SYSTEM V RETURN VALUES

nice() returns the new nice value on success. On failure, it returns –1 and sets **errno** to indicate the error.

ERRORS

The nice value is not changed if:

EACCES The value of *incr* specified was negative, and the effective user ID is not super-user.

SYSTEM V ERRORS

The nice value is not changed if:

EPERM The value of *incr* specified was negative, or greater than 40, and the effective user ID is not super-user.

SEE ALSO

nice(1), **fork**(2V), **getpriority**(2), **pstat**(8), **renice**(8)

NAME

nl_langinfo – language information

SYNOPSIS

#include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo(item)
nl_item item;

DESCRIPTION

nl_langinfo( ) returns a pointer to a null-terminated string containing information relevant to a particular language or cultural area defined in the program's locale. The manifest constant names and values of *item* are defined in **<langinfo.h>** . For example:

nl_langinfo(ABDAY_1);

would return a pointer to the string 'Dom' if the identified language was Portuguese, and 'Sun' if the identified language was English.

RETURN VALUES

In a locale where *langinfo* data is not defined, **nl_langinfo()** returns a pointer to the corresponding string in the "C" locale. In all locales **nl_langinfo( )** returns a pointer to an empty string if *item* contains an invalid setting.

SEE ALSO

setlocale(3V), **environ**(5V)

NAME

nlist – get entries from symbol table

SYNOPSIS

#include <nlist.h>

int nlist(filename, nl)
char *filename;
struct nlist *nl;

DESCRIPTION

nlist() examines the symbol table from the executable image whose name is pointed to by *filename*, and selectively extracts a list of values and puts them in the array of **nlist()** structures pointed to by *nl*. The name list pointed to by *nl* consists of an array of structures containing names, types and values. The *n_name* field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a null string) in the *n_name* field. For each entry in *nl*, if the named symbol is present in the executable image's symbol table, its value and type are placed in the *n_value* and *n_type* fields. If a symbol cannot be located, the corresponding *n_type* field of *nl* is set to zero.

RETURN VALUES

On success, **nlist()** returns the number of symbols that were not located in the symbol table. On failure, it returns −1 and sets all of the *n_type* fields in members of the array pointed to by *nl* to zero.

SYSTEM V RETURN VALUES

nlist() returns 0 on success.

SEE ALSO

a.out(5), **coff**(5)

NOTES

On Sun-2, Sun-3, and Sun-4 systems, type entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

On Sun386i systems, the type entries may be zero even when the name list succeeded, but the value entries will be zero only when the file cannot be read or does not contain a valid name list. Therefore, on Sun386i systems, the value entry can be used to determine whether the command succeeded.

NAME

on_exit – name termination handler

SYNOPSIS

int on_exit(procp, arg)
void (*procp)();
caddr_t arg;

DESCRIPTION

on_exit() names a routine to be called after a program calls **exit**(3) or returns normally, and before its process terminates. The routine named is called as

(*procp)(status, arg);

where *status* is the argument with which **exit()** was called, or zero if *main* returns. Typically, *arg* is the address of an argument vector to (**procp*), but may be an integer value. Several calls may be made to **on_exit**, specifying several termination handlers. The order in which they are called is the reverse of that in which they were given to **on_exit**.

SEE ALSO

gprof(1), **tcov**(1), **exit**(3)

DIAGNOSTICS

on_exit() returns zero normally, or nonzero if the procedure name could not be stored.

NOTES

This call is specific to the SunOS operating system and should not be used if portability is a concern.

Standard I/O exit processing is always done last.

NAME

pause – stop until signal

SYNOPSIS

int pause()

DESCRIPTION

pause() never returns normally. It is used to give up control while waiting for a signal from **kill**(2V) or an interval timer, see **getitimer**(2). Upon termination of a signal handler started during a pause, **pause()** will return.

RETURN VALUES

When it returns, **pause()** returns −1.

ERRORS

When it returns, **pause()** sets **errno** to:

EINTR	A signal is caught by the calling process and control is returned from the signal-catching function.

SEE ALSO

kill(2V), **getitimer**(2), **select**(2), **sigpause**(2V)

NAME

perror, errno – system error messages

SYNOPSIS

```
void perror(s)
char *s;

#include <errno.h>

int sys_nerr;
char *sys_errlist[ ];
int errno;
```

DESCRIPTION

perror( ) produces a short error message on the standard error describing the last error encountered during a call to a system or library function. If *s* is not a NULL pointer and does not point to a null string, the string it points to is printed, followed by a colon, followed by a space, followed by the message and a NEWLINE. If *s* is a NULL pointer or points to a null string, just the message is printed, followed by a NEWLINE. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable **errno** (see **intro**(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings **sys_errlist** is provided; **errno** can be used as an index in this table to get the message string without the newline. **sys_nerr** is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2), **psignal**(3)

NAME

plock – lock process, text, or data segment in memory

SYNOPSIS

#include <sys/lock.h>

int plock(op)
int op;

DESCRIPTION

plock() allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. **plock()** also allows these segments to be unlocked. The effective user ID of the calling process must be super-user to use this call. *op* specifies the following:

PROCLOCK	lock text and data segments into memory (process lock)
TXTLOCK	lock text segment into memory (text lock)
DATLOCK	lock data segment into memory (data lock)
UNLOCK	remove locks

RETURN VALUES

plock() returns:

0	on success.
–1	on failure and sets **errno** to indicate the error.

ERRORS

EAGAIN	Not enough memory.
EINVAL	*op* is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process.
	op is equal to **TXTLOCK** and a text lock, or a process lock already exists on the calling process.
	op is equal to **DATLOCK** and a data lock, or a process lock already exists on the calling process.
	op is equal to **UNLOCK** and no type of lock exists on the calling process.
EPERM	The effective user ID of the calling process is not super-user.

SEE ALSO

execve(2V), **exit**(2V), **fork**(2V)

NAME

plot, openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl – graphics interface

SYNOPSIS

openpl()

erase()

label(s)
char s[ ];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, x0, y0, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s)
char s[ ];

space(x0, y0, x1, y1)

closepl()

AVAILABILITY

These routines are available with the *Graphics* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

LP These subroutines generate graphic output in a relatively device-independent manner. See **plot**(5) for a description of their effect. **openpl()** must be used before any of the others to open the device for writing. **closepl()** flushes the output.

String arguments to **label()** and **linemod()** are null-terminated and do not contain NEWLINE characters.

Various flavors of these functions exist for different output devices. They are obtained by the following **ld**(1) options:

–lplot	device-independent graphics stream on standard output for **plot**(1G) filters
–l300	GSI 300 terminal
–l300s	GSI 300S terminal
–l450	GSI 450 terminal
–l4014	Tektronix 4014 terminal
–lplotaed	AED 512 color graphics terminal
–lplotbg	BBN bitgraph graphics terminal
–lplotdumb	Dumb terminals without cursor addressing or line printers
–lplotgigi	DEC Gigi terminals
–lplot2648	Hewlett Packard 2648 graphics terminal
–lplot7221	Hewlett Packard 7221 graphics terminal
–lplotimagen	Imagen laser printer (default 240 dots-per-inch resolution).

FILES

/usr/lib/libplot.a
/usr/lib/lib300.a
/usr/lib/lib300s.a
/usr/lib/lib450.a
/usr/lib/lib4014.a
/usr/lib/libplotaed.a
/usr/lib/libplotbg.a
/usr/lib/libplotdumb.a
/usr/lib/libplotgigi.a
/usr/lib/libplot2648.a
/usr/lib/libplot7221.a
/usr/lib/libplotimagen.a

SEE ALSO

graph(1G), **ld**(1), **plot**(1G), **plot**(5)

NAME

popen, pclose – open or close a pipe (for I/O) from or to a process

SYNOPSIS

#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;

DESCRIPTION

The arguments to **popen**() are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either **r** for reading or **w** for writing. **popen**() creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is **w**, by writing to the file stream; and one can read from the standard output of the command, if the I/O mode is **r**, by reading from the file stream.

A stream opened by **popen**() should be closed by **pclose**(), which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type **r** command may be used as an input filter, reading its standard input (which is also the standard output of the process doing the **popen**()) and providing filtered input on the stream, and a type **w** command may be used as an output filter, reading a stream of output written to the stream process doing the **popen**() and further filtering it and writing it to its standard output (which is also the standard input of the process doing the **popen**()).

popen() always calls **sh**(1), never **csh**(1).

SEE ALSO

csh(1), **sh**(1), **pipe**(2V), **wait**(2V), **fclose**(3V), **fopen**(3V), **system**(3)

DIAGNOSTICS

popen() returns a NULL pointer if the pipe or process cannot be created, or if it cannot allocate as much memory as it needs.

pclose() returns –1 if stream is not associated with a '**popen**ed' command.

BUGS

If the original and '**popened**' processes concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Similar problems with an output filter may be forestalled by careful buffer flushing, for instance, with **fflush**(); see **fclose**(3V).

NAME

pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set, pmap_unset, xdr_pamp, xdr_pmaplist – library routines for RPC bind service

DESCRIPTION

These routines allow client C programs to make procedure calls to the RPC binder service. **portmap**(1) maintains a list of mappings between programs and their universal addresses.

Routines

#include <rpc/rpc.h>

struct pmaplist * pmap_getmaps(addr)
struct sockaddr_in *addr;

Return a list of the current RPC program-to-address mappings on the host located at IP address **addr*. This routine returns NULL if the remote portmap service could not be contacted. The command '**rpcinfo –p**' uses this routine (see **rpcinfo**(8C)).

u_short pmap_getport(addr, prognum, versnum, protocol)
struct sockaddr_in *addr;
u_long prognum, versnum, protocol;

Return the port number on which waits a service that supports program number *prognum*, version *versnum*, and speaks the transport protocol *protocol*. The address is returned in *addr*, which should be preallocated. The value of *protocol* can be either **IPPROTO_UDP** or **IPPROTO_TCP**. A return value of zero means that the mapping does not exist or that the RPC system failed to contact the remote **portmap** service. In the latter case, the global variable **rpc_createer** (see **rpc_clnt_create**(3N)) contains the RPC status. If the requested version number is not registered, but at least a version number is registered for the given program number, the call returns a port number. Note: **pmap_getport()** returns the port number in host byte order. Some other network routines may require the port number in network byte order. For example, if the port number is used as part of the **sockaddr_in** structure, then it should be converted to network byte order using **htons**(3N).

enum clnt_stat pmap_rmtcall(addr, prognum, versnum, procnum, inproc, in, outproc, out, timeout, portp)
struct sockaddr_in *addr;
u_long prognum, versnum, procnum;
char *in, *out;
xdrproc_t inproc, outproc;
struct timeval timeout;
u_long *portp;

Request that the **portmap** on the host at IP address **addr* make an RPC on the behalf of the caller to a procedure on that host. **portp* is modified to the program's port number if the procedure succeeds. The definitions of other parameters are discussed in **callrpc()** and **clnt_call()** (see **rpc_clnt_calls**(3N)).

Warning: If the requested remote procedure is not registered with the remote **portmap** then no error response is returned and the call times out. Also, no authentication is done.

bool_t pmap_set(prognum, versnum, protocol, port)
u_long prognum, versnum;
int protocol;
u_short port;

Registers a mapping between the triple [*prognum*,*versnum*,*protocol*] and *port* on the local machine's **portmap** service. The value of *protocol* can be either **IPPROTO_UDP** or **IPPROTO_TCP**. This routine returns TRUE if it succeeds, FALSE otherwise. It is called by servers to register themselves with the local **portmap**. Automatically done by **svc_register()**.

bool_t pmap_unset(prognum, versnum)
u_long prognum, versnum;

Deregisters all mappings between the triple [*prognum*,*versnum*,*] and ports on the local machine's **portmap** service. It is called by servers to deregister themselves with the local **portmap.** This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_pmap(xdrs, regp)
XDR *xdrs;
struct pmap *regp;

Used for creating parameters to various **portmap** procedures, externally. This routine is useful for users who wish to generate these parameters without using the **pmap** interface. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_pmaplist(xdrs, rp)
XDR *xdrs;
struct pmaplist **rp;

Used for creating a list of port mappings, externally. This routine is useful for users who wish to generate these parameters without using the **pmap** interface. This routine returns TRUE if it succeeds, FALSE otherwise.

SEE ALSO

rpc(3N), **portmap**(8C), **rpcinfo**(8C)

NAME

printf, fprintf, sprintf – formatted output conversion

SYNOPSIS

#include <stdio.h>

int printf(format [**, arg** . . .])
char *format;

int fprintf(stream, format [**, arg** . . .])
FILE *stream;
char *format;

char *sprintf(s, format [**, arg** . . .])
char *s, *format;

SYSTEM V SYNOPSIS

The routines above are available as shown, except:

int sprintf(s, format [**, arg** . . .])
char *s, *format;

The following are provided for XPG2 compatibility:

#define nl_printf printf
#define nl_fprintf fprintf
#define nl_sprintf sprintf

DESCRIPTION

printf( ) places output on the standard output stream **stdout. fprintf( )** places output on the named output **stream. sprintf( )** places "output", followed by the null character (**\0**), in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available.

Each of these functions converts, formats, and prints its *args* under control of the *format*. The *format* is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of zero or more *args*. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

Each conversion specification is introduced by either the **%** character or by the character sequence ***%digit*$**, after which the following appear in sequence:

- Zero or more *flags*, which modify the meaning of the conversion specification.
- An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–', described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.
- A *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.
- An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.
- A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a '–' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an "alternate form". For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x or X** conversion, a non-zero result will have **0x or 0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,i,o,p,u,x,X
The integer *arg* is converted to signed decimal (**d** or **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x**, **p**, and **X**), respectively; the letters **abcdef** are used for **x** and **p** conversion and the letters ABCDEF for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

f The float or double *arg* is converted to decimal notation in the style "[–]ddd.ddd" where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e,E The float or double *arg* is converted in the style "[–]d.ddde±ddd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

g,G The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** or **E** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The **e**, **E**, **f**, **g**, and **G** formats print IEEE indeterminate values (infinity or not-a-number) as "Infinity" or "NaN" respectively.

c The character *arg* is printed.

s The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (\0) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for *arg* will yield undefined results.

n The argument *arg* is a pointer to an integer into which is written the number of characters written to the output so far by this call to one of the **printf( )** functions. No argument is converted.

% Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated by **printf( )** and **fprintf( )** are printed as if **putc**(3S) had been called.

All forms of the **printf( )** functions allow for the insertion of a language dependent radix character in the output string. The radix character is defined by the program's locale (category **LC_NUMERIC**). In the **"C"** locale, or in a locale where the radix character is not defined, the radix character defaults to '**.**'.

Conversions can be applied to the *n*th argument in the argument list, rather than the next unused argument. In this case, the conversion character **%** is replaced by the sequence ***%digit*$**, where *digit* is a decimal integer *n* in the range [1,9], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages.

In format strings containing the ***%digit*$** form of a conversion specification, a field width or precision may be indicated by the sequence ***digit*$**, where *digit* is a decimal integer in the range [1,9] giving the position in the argument list of an integer *arg* containing the field width or precision.

The format string can contain either numbered argument specifications (that is, ***%digit*$** and ***digit*$**), or unnumbered argument specifications (that is **%** and *****), but not both. The results of mixing numbered and unnumbered specifications is undefined. When numbered argument specifications are used, specifying the *n*th argument requires that all the leading arguments, from the first to the (*n*-1)th be specified in the format string.

SYSTEM V DESCRIPTION

XPG2 requires that **nl_printf**, **nl_fprintf** and **nl_sprintf** be defined as **printf**, **fprintf** and **sprintf**, respectively for backward compatibility

RETURN VALUES

On success, **printf( )** and **fprintf( )** return the number of characters transmitted, excluding the null character. On failure, they return EOF.

sprintf( ) returns *s*.

SYSTEM V RETURN VALUES

On success, **sprintf( )** returns the number of characters transmitted, excluding the null character. On failure, it returns EOF.

EXAMPLES

```
printf(format, weekday, month, day, hour, min);
```

In American usage, *format* could be a pointer to the string:

```
"%s, %s %d, %d:%.2d\n"
```

producing the message:

```
Sunday, July 3,10:02
```

Whereas for German usage, *format* could be a pointer to the string:

```
"%1$s, %3$d.%2$s,%4$d:%5$.2d\n"
```

producing the message:

```
Sonntag, 3.Juli,10:02
```

To print π to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1.0));
```

SEE ALSO

econvert(3), **putc**(3S), **scanf**(3V), **setlocale**(3V), **varargs**(3), **vprintf**(3V)

BUGS

Very wide fields (>128 characters) fail.

NAME

prof – profile within a function

SYNOPSIS

#define MARK
#include <prof.h>

void MARK (name)

DESCRIPTION

MARK introduces a mark called *name* that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file **<prof.h>** is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, such as:

```
cc –p –DMARK foo.c
```

If MARK is not defined, the MARK (**name**) statements may be left in the source files containing them and will be ignored.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with MARK defined on the command line, the marks are ignored.

```
#include <prof.h>
func( )
{
        int i, j;
        .
        .
        .
        MARK (loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK (loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

prof(1), **profil**(2), **monitor**(3)

NAME

psignal, sys_siglist – system signal messages

SYNOPSIS

psignal(sig, s)
unsigned sig;
char *s;

char *sys_siglist[ ];

DESCRIPTION

psignal() produces a short message on the standard error file describing the indicated signal. First the argument string *s* is printed, then a colon, then the name of the signal and a NEWLINE. Most usefully, the argument string is the name of the program which incurred the signal. The signal number should be from among those found in **<signal.h>**.

To simplify variant formatting of signal names, the vector of message strings **sys_siglist()** is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define **NSIG** defined in **<signal.h>** is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO

perror(3), **signal**(3V)

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

int putchar(c)
char c;

int fputc(c, stream)
char c;
FILE *stream;

int putw(w, stream)
int w;
FILE *stream;
```

DESCRIPTION

putc() writes the character *c* onto the standard I/O output stream *stream* (at the position where the file pointer, if defined, is pointing). It returns the character written.

putchar(c) is defined as **putc(c, stdout)**. **putc()** and **putchar()** are macros.

fputc() behaves like **putc()**, but is a function rather than a macro. **fputc()** runs more slowly than **putc()**, but it takes less space per invocation and its name can be passed as an argument to a function.

putw() writes the C **int** (word) **w** to the standard I/O output stream *stream* (at the position of the file pointer, if defined). The size of a word is the size of an integer and varies from machine to machine. **putw()** neither assumes nor causes special alignment in the file.

Output streams are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a NEWLINE character is written or terminal input is requested). **setbuf**(3V), **setbuffer()**, or **setvbuf()** may be used to change the stream's buffering strategy.

SEE ALSO

fclose(3V), **ferror**(3V), **fopen**(3V), **fread**(3S), **getc**(3V), **printf**(3V), **puts**(3S), **setbuf**(3V)

DIAGNOSTICS

On success, **putc()**, **fputc()**, and **putchar()** return the value that was written. On error, those functions return the constant EOF. **putw()** returns **ferror(stream)**, so that it returns 0 on success and 1 on failure.

BUGS

Because it is implemented as a macro, **putc()** treats a *stream* argument with side effects improperly. In particular, **putc(c, *f++);** does not work sensibly. **fputc()** should be used instead.

Errors can occur long after the call to **putc()**.

Because of possible differences in word length and byte ordering, files written using **putw()** are machine-dependent, and may not be read using **getw()** on a different processor.

NAME

putenv – change or add value to environment

SYNOPSIS

int putenv(string)
char *string;

DESCRIPTION

string points to a string of the form '*name=value*' **putenv**( ) makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* becomes part of the environment, so altering the string will change the environment. The space used by *string* is no longer used once a new string-defining *name* is passed to **putenv**( ).

SEE ALSO

execve(2V), **getenv**(3V), **malloc**(3V), **environ**(5V)

DIAGNOSTICS

putenv( ) returns non-zero if it was unable to obtain enough space using **malloc**(3V) for an expanded environment, otherwise zero.

WARNINGS

putenv( ) manipulates the environment pointed to by *environ*, and can be used in conjunction with **getenv**( ). However, *envp* (the third argument to *main*) is not changed.

This routine uses **malloc**(3V) to enlarge the environment.

After **putenv**() is called, environmental variables are not in alphabetical order.

A potential error is to call **putenv**( ) with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

NAME

putpwent – write password file entry

SYNOPSIS

#include <pwd.h>

int putpwent(p, f)
struct passwd *p;
FILE *f;

DESCRIPTION

putpwent( ) is the inverse of **getpwent**(3V). Given a pointer to a passwd structure created by **getpwent( )** (or **getpwuid( )** or **getpwnam**), **putpwent( )** writes a line on the stream *f*, which matches the format of lines in the password file **/etc/passwd**.

FILES

/etc/passwd

SEE ALSO

getpwent(3V)

DIAGNOSTICS

putpwent( ) returns non-zero if an error was detected during its operation, otherwise zero.

WARNING

The above routine uses **<stdio.h>**, which increases the size of programs, not otherwise using standard I/O, more than might be expected.

BUGS

This routine is of limited utility, since most password files are maintained as Network Information Service (NIS) files, and cannot be updated with this routine.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

puts, fputs – put a string on a stream

SYNOPSIS

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

puts( ) writes the null-terminated string pointed to by *s*, followed by a NEWLINE character, to the standard output stream **stdout**.

fputs( ) writes the null-terminated string pointed to by *s* to the named output stream.

Neither function writes the terminal null character.

DIAGNOSTICS

Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.

NOTES

puts( ) appends a NEWLINE while **fputs**( ) does not.

SEE ALSO

ferror(3V), **fopen**(3V), **fread**(3S), **printf**(3V), **putc**(3S)

NAME

pwdauth, grpauth – password authentication routines

SYNOPSIS

int pwdauth(user, password)
char *user;
char *password;

int grpauth(group, password)
char *group;
char *password;

DESCRIPTION

pwdauth() and **grpauth()** determine whether the given guess at a *password* is valid for the given *user* or *group*. If the *password* is valid, the functions return 0.

A *password* is valid if the password when encrypted matches the encrypted password in the appropriate file. For **pwdauth()**, if the **password.adjunct** file exists, the encrypted password will be in either the local or the Network Information Service (NIS) version of that file. Otherwise, either the local or NIS **passwd** file will be used. For **grpauth()**, the **group.adjunct** file (if it exists) or the **group** file (otherwise) will be checked on the local machine and then using the NIS service. In all cases, the local files will be checked before the NIS files. Also, if the adjunct files exist, the main file will never be used for authentication even if they include encrypted passwords.

Both **pwdauth()** and **grpauth()** interface to the authentication daemon, **rpc.pwdauthd**, to do the checking of the adjunct files. This daemon must be running on any system that provides password authentication.

FILES

/etc/passwd
/etc/group

SEE ALSO

getgraent(3), **getgrent**(3V), **getpwaent**(3), **getpwent**(3V), **pwdauthd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

qsort – quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)();
```

DESCRIPTION

qsort() is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

base points to the element at the base of the table. *nel* is the number of elements in the table. *width* is the size, in bytes, of each element in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. As the function must return an integer less than, equal to, or greater than zero, so must the first argument to be considered be less than, equal to, or greater than the second.

NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The order in the output of two items which compare as equal is unpredictable.

SEE ALSO

sort(1V), **bsearch**(3), **lsearch**(3), **string**(3)

EXAMPLE

The following program sorts a simple array:

```
static	int intcompare(i,j)
int *i, *j;
{
	return(*i – *j);
}

main()
{
	int a[10];
	int i;

	a[0] = 9;
	a[1] = 8;
	a[2] = 7;
	a[3] = 6;
	a[4] = 5;
	a[5] = 4;
	a[6] = 3;
	a[7] = 2;
	a[8] = 1;
	a[9] = 0;

	qsort(a,10,sizeof(int),intcompare)

	for (i=0; i<10; i++) printf(" %d",a[i]);
	printf("\n");
}
```

NAME

rand, srand – simple random number generator

SYNOPSIS

srand(seed)
int seed;

rand( )

DESCRIPTION

rand( ) uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

srand( ) can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

SYSTEM V DESCRIPTION

rand( ) returns successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

SEE ALSO

drand48(3), **random**(3)

NOTES

The spectral properties of **rand( )** leave a great deal to be desired. **drand48**(3) and **random**(3) provide much better, though more elaborate, random-number generators.

BUGS

The low bits of the numbers generated are not very random; use the middle bits. In particular the lowest bit alternates between 0 and 1.

NAME

random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS

long random( )

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;

DESCRIPTION

random( ) uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16\times(2^{31}-1)$.

random/srandom have (almost) the same calling sequence and initialization properties as **rand/srand.** The difference is that **rand**(3V) produces a much less random sequence — in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by **random**( ) are usable. For example,

random( )&01

will produce a random binary value.

Unlike **srand**, **srandom**( ) does not return the old seed; the reason for this is that the amount of state information used is much more than a single word. (Two other routines are provided to deal with restarting/changing random number generators). Like **rand**(3V), however, **random**( ) will by default produce a sequence of numbers that can be duplicated by calling **srandom**( ) with *1* as the seed.

The **initstate**( ) routine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by **initstate**( ) to decide how sophisticated a random number generator it should use — the more state, the better the random numbers will be. (Current ''optimal'' values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error). The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. **initstate**( ) returns a pointer to the previous state information array.

Once a state has been initialized, the **setstate**( ) routine provides for rapid switching between states. **setstate**( ) returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to **initstate**( ) or **setstate**( ).

Once a state array has been initialized, it may be restarted at a different point either by calling **initstate**( ) (with the desired seed, the state array, and its size) or by calling both **setstate**( ) (with the state array) and **srandom**( ) (with the desired seed). The advantage of calling both **setstate**( ) and **srandom**( ) is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than 2^{69}, which should be sufficient for most purposes.

SEE ALSO

rand(3V)

EXAMPLES

```
/* Initialize and array and pass it in to initstate. */

static long state1[32] = {
        3,
        0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
        0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
        0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
        0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
        0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
        0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
        0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
        0xf5ad9d0e, 0x8999220b, 0x27fb47b9
        };

main()
{
        unsigned seed;
        int n;

        seed = 1;
        n = 128;
        initstate(seed, (char *) state1, n);

        setstate(state1);
        printf("%d\n",random());
}
```

DIAGNOSTICS

If **initstate**( ) is called with less than 8 bytes of state information, or if **setstate**( ) detects that the state information has been garbled, error messages are printed on the standard error output.

WARNINGS

initstate( ) casts *state* to (**long** *), so *state* must be long-aligned. If it is not long-aligned, on some architectures the program will dump core.

BUGS

random( ) is only 2/3 as fast as **rand**(3V).

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

int rcmd(ahost, inport, locuser, remuser, cmd, fd2p)
char **ahost;
unsigned short inport;
char *locuser, *remuser, *cmd;
int *fd2p

int rresvport(port)
int *port;

ruserok(rhost, super-user, ruser, luser)
char *rhost;
int super-user;
char *ruser, *luser;

DESCRIPTION

rcmd( ) is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. **rresvport( )** is a routine which returns a descriptor to a socket with an address in the privileged port space. **ruserok( )** is a routine used by servers to authenticate clients requesting service with **rcmd**. All three functions are present in the same file and are used by the **rshd**(8C) server (among others).

rcmd( ) looks up the host **ahost* using **gethostbyname** (see **gethostent**(3N)), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the connection succeeds, a socket in the Internet domain of type **SOCK_STREAM** is returned to the caller, and given to the remote command as its standard input (file descriptor 0) and standard output (file descriptor 1). If *fd2p* is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (file descriptor 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (file descriptor 2) of the remote command will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in **rshd**(8C).

The **rresvport( )** routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by **rcmd( )** and several other routines. Privileged Internet ports are those in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

ruserok( ) takes a remote host's name, as returned by a **gethostbyaddr** (see **gethostent**(3N)) routine, two user names and a flag indicating whether the local user's name is that of the super-user. It then checks the files **/etc/hosts.equiv** and, possibly, **.rhosts** in the local user's home directory to see if the request for service is allowed. A 0 is returned if the machine name is listed in the **/etc/hosts.equiv** file, or the host and remote user name are found in the **.rhosts** file; otherwise **ruserok( )** returns –1. If the super-user flag is 1, the checking of the **/etc/hosts.equiv** file is bypassed.

FILES

/etc/hosts.equiv
.rhosts

SEE ALSO

rlogin(1C), **rsh**(1C), **intro**(2), **gethostent**(3N), **rexec**(3N), **rexecd**(8C), **rlogind**(8C), **rshd**(8C)

DIAGNOSTICS

rcmd() returns a valid socket descriptor on success. It returns −1 on error and prints a diagnostic message on the standard error.

rresvport() returns a valid, bound socket descriptor on success. It returns −1 on error with the global value **errno** set according to the reason for failure. The error code EAGAIN is overloaded to mean "All network ports in use."

NAME

realpath – return the canonicalized absolute pathname

SYNOPSIS

#include <sys/param.h>

char *realpath(path, resolved_path)
char *path;
char resolved_path[MAXPATHLEN];

DESCRIPTION

realpath() expands all symbolic links and resolves references to '/./', '/../' and extra '/' characters in the null terminated string named by *path* and stores the canonicalized absolute pathname in the buffer named by *resolved_path*. The resulting path will have no symbolic links components, nor any '/./' or '/../' components.

RETURN VALUES

realpath() returns a pointer to the *resolved_path* on success. On failure, it returns NULL, sets **errno** to indicate the error, and places in *resolved_path* the absolute pathname of the *path* component which could not be resolved.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
EFAULT	*resolved_path* extends outside the process's allocated address space.
ELOOP	Too many symbolic links were encountered in translating *path*.
EINVAL	*path* or *resolved_path* was NULL.
EIO	An I/O error occurred while reading from or writing to the file system.
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} (see **sysconf**(2V)) while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The named file does not exist.

SEE ALSO

readlink(2), **getwd**(3)

WARNINGS

It indirectly invokes the **readlink(2)** system call and **getwd(3)** library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

NAME

regex, re_comp, re_exec – regular expression handler

SYNOPSIS

char *re_comp(s)
char *s;

re_exec(s)
char *s;

DESCRIPTION

re_comp( ) compiles a string into an internal form suitable for pattern matching. **re_exec( )** checks the argument string against the last string passed to **re_comp( )**.

re_comp( ) returns a NULL pointer if the string *s* was compiled successfully; otherwise a string containing an error message is returned. If **re_comp( )** is passed 0 or a null string, it returns without changing the currently compiled regular expression.

re_exec( ) returns 1 if the string *s* matches the last compiled regular expression, 0 if the string *s* failed to match the last compiled regular expression, and –1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both **re_comp( )** and **re_exec( )** may have trailing or embedded NEWLINE characters; they are terminated by null characters. The regular expressions recognized are described in the manual entry for **ed**(1), given the above difference.

SEE ALSO

ed(1), **ex**(1), **grep**(1V)

DIAGNOSTICS

re_exec( ) returns –1 for an internal error.

re_comp( ) returns one of the following strings if an error occurs:

No previous regular expression

Regular expression too long

unmatched \(

missing]

too many \(\) pairs

unmatched \)

NAME

regexp – regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step(string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

DESCRIPTION

This page describes general-purpose regular expression matching routines.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the '**#include <regexp.h>**' statement. These macros are used by the *compile* routine.

GETC()
: Return the value of the next character in the regular expression pattern. Successive calls to **GETC()** should return successive characters of the regular expression.

PEEKC()
: Return the next character in the regular expression. Successive calls to **PEEKC()** should return the same character, which should also be the next character returned by **GETC()**.

UNGETC(*c*)
: Returns the argument *c* by the next call to **GETC()** or **PEEKC()**. No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by **GETC()**. The value of the macro **UNGETC(*c*)** is always ignored.

RETURN(*pointer*)
: This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs that have memory allocation to manage.

ERRORS

ERROR(*val*)
: This is the abnormal return from the **compile()** routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	''\ digit'' out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.

44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression too long.

The syntax of the **compile()** routine is as follows:

compile(instring, expbuf, endbuf, eof)

The first parameter *instring* is never used explicitly by the **compile()** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the **INIT()** declaration (see below). Programs that call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to **ERROR(50)** is made.

The parameter *eof* is the character that marks the end of the regular expression. For example, in an editor like **ed**(1), this character would usually a '/'.

Each program that includes this file must have a **#define** statement for **INIT()**. This definition will be placed right after the declaration for the function **compile()** and '{' (opening curly brace). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for **GETC()**, **PEEKC()**, and **UNGETC()**. Otherwise it can be used to declare external variables that might be used by **GETC()**, **PEEKC()**, and **UNGETC()**. See the example below of the declarations taken from **grep**(1V).

There are other functions in this file that perform actual regular expression matching, one of which is the function **step()**. The call to **step()** is as follows:

step(string, expbuf)

The first parameter to **step()** is a pointer to a string of characters to be checked for a match. This string should be null-terminated

The second parameter *expbuf* is the compiled regular expression that was obtained by a call of the function *compile*.

The function **step()** returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to **step()**. The variable set in **step()** is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function **advance()**, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, **loc1** will point to the first character of *string* and **loc2** will point to the null character at the end of *string*.

step() uses the external variable **circf** which is set by **compile()** if the regular expression begins with '^'. If this is set then **step()** will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of **circf** should be saved for each compiled expression and **circf** should be set to that saved value before each call to **step()**.

The function **advance()** is called from **step()** with the same arguments as **step()**. The purpose of **step()** is to step through the *string* argument and call **advance()** until **advance()** returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, **step()** need not be called; simply call **advance()**.

When **advance()** encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance()** will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer **locs** is equal to the point in the string at sometime during the backing up process, **advance()** will break out of the loop that backs up and will return zero. This could be used by an editor like **ed**(1) or **sed**(1V) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables **sed** and **nbra** are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls could look in a command like **grep**(1V):

```
#define INIT        register char *sp = instring;
#define GETC() (*sp++)
#define PEEKC()         (*sp)
#define UNGETC(c)       (--sp)
#define RETURN(c)       return;
#define ERROR(c)        regerr()

#include <regexp.h>
...
                (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
                if (step(linebuf, expbuf))
                                succeed ();
```

SEE ALSO

ed(1), **grep**(1V), **sed**(1V)

BUGS

The handling of **circf** is difficult.

NAME

resolver, res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
u_char *exp_dn, *comp_dn;
int length;
u_char **dnptrs, **lastdnptr;

dn_expand(msg, msglen, comp_dn, exp_dn, length)
u_char *msg, *eomorig, *comp_dn, exp_dn;
int length;
```

DESCRIPTION

These routines are used for making, sending and interpreting packets to Internet domain name servers. You can link a program with the resolver library using the **–lresolv** argument on the linking command line.

Global information that is used by the resolver routines is kept in the variable *_res*. Most of the values have reasonable defaults and can be ignored. Options are a simple bit mask and are OR'ed in to enable. Options stored in *_res.options* are defined in **<resolv.h>** and are as follows.

RES_INIT True if the initial name server address and default domain name are initialized (that is, **res_init()** has been called).

RES_DEBUG Print debugging messages.

RES_AAONLY Accept authoritative answers only. **res_send()** continues until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC Use TCP connections for queries instead of UDP.

RES_STAYOPEN Used with **RES_USEVC** to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC Unused currently (ignore truncation errors, that is, do not retry with TCP).

RES_RECURSE Set the recursion desired bit in queries. This is the default. **res_send()** does not do iterative queries and expects the name server to handle recursion.

RES_DEFNAMES Append the default domain name to single label queries. This is the default.

RES_DNSRCH Search up the domain tree from the default domain, in all but the top level. This is the default.

res_init( ) reads the initialization file to get the default domain name and the Internet addresses of the initial name servers. If no **nameserver** line exists, the host running the resolver is tried. **res_mkquery( )** makes a standard query message and places it in *buf*. **res_mkquery( )** returns the size of the query or −1 if the query is larger than *buflen*. *op* is usually **QUERY** but can be any of the query types defined in **<nameser.h>**. *dname* is the domain name. If *dname* consists of a single label and the **RES_DEFNAMES** flag is enabled (the default), *dname* is appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable **LOCALDOMAIN**. *newrr* is currently unused but is intended for making update messages.

res_send( ) sends a query to name servers and returns an answer. It calls **res_init( )** if **RES_INIT** is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or −1 if there were errors.

dn_expand( ) Expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or −1 if there was an error.

dn_comp( ) Compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or −1 if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to the beginning of the message and the list ends with NULL. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by **dn_comp( )** as the name is compressed. If *dnptr* is NULL, do not try to compress names. If *lastdnptr* is NULL, do not update the list.

FILES

/etc/resolv.conf see **resolv.conf**(5)
/usr/lib/libresolv.a

SEE ALSO

resolv.conf(5), **named**(8C)

System and Network Administration

NOTES

/usr/lib/libresolv.a is necessary for compiling programs.

NAME

rexec – return stream to a remote command

SYNOPSIS

rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;

DESCRIPTION

rexec() looks up the host **ahost* using **gethostbyname()** (see **gethostent**(3N)), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's **.netrc** file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port **inport** specifies which well-known DARPA Internet port to use for the connection; it will normally be the value returned from the call '**getservbyname("exec", "tcp")**' (see **getservent**(3N)). The protocol for connection is described in detail in **rexecd**(8C).

If the call succeeds, a socket of type **SOCK_STREAM** is returned to the caller, and given to the remote command as its standard input and standard output. If *fd2p* is non-zero, then a auxiliary channel to a control process will be setup, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (unit 2 of the remote command) will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

gethostent(3N), **getservent**(3N), **rcmd**(3N), **rexecd**(8C)

BUGS

There is no way to specify options to the **socket()** call that **rexec()** makes.

NAME

rpc – library routines for remote procedure calls

SYNOPSIS AND DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

All RPC routines require the header **<rpc/rpc.h>** to be included.

The RPC routines have been grouped by usage on the following man pages.

portmap(3N) Library routines for the RPC bind service, **portmap**(8C). The routines documented on this page include:

- **pmap_getmaps()**
- **pmap_getport()**
- **pmap_rmtcall()**
- **pmap_set()**
- **pmap_unset()**
- **xdr_pmap()**
- **xdr_pmaplist()**

rpc_clnt_auth(3N) Library routines for client side remote procedure call authentication. The routines documented on this page include:

- **auth_destroy()**
- **authnone_create()**
- **authunix_create()**
- **authunix_create_default()**

rpc_clnt_calls(3N) Library routines for client side calls. The routines documented on this page include:

- **callrpc()**
- **clnt_broadcast()**
- **clnt_call()**
- **clnt_freeres()**
- **clnt_geterr()**
- **clnt_perrno()**
- **clnt_perror()**
- **clnt_sperrno()**
- **clnt_sperror()**

rpc_clnt_create(3N) Library routines for dealing with the creation and manipulation of CLIENT handles. The routines documented on this page include:

- **clnt_control()**
- **clnt_create()**
- **clnt_create_vers()**
- **clnt_destroy()**
- **clnt_pcreateerror()**
- **clntraw_create()**
- **clnt_spcreateerror()**
- **clnttcp_create()**
- **clntudp_bufcreate()**
- **clntudp_create()**
- **rpc_createrr()**

rpc_svc_calls(3N) Library routines for registerring servers. The routines documented on this page include:

- **registerrpc( )**
- **svc_register( )**
- **svc_unregister( )**
- **xprt_register( )**
- **xprt_unregister( )**

rpc_svc_create(3N) Library routines for dealing with the creation of server side handles. The routines documented on this page include:

- **svc_destroy( )**
- **svcfd_create( )**
- **svcraw_create( )**
- **svctcp_create( )**
- **svcudp_bufcreate( )**

rpc_svc_err(3N) Library routines for server side remote procedure call errors. The routines documented on this page include:

- **svcerr_auth( )**
- **svcerr_decode( )**
- **svcerr_noproc( )**
- **svcerr_noprog( )**
- **svcerr_progvers( )**
- **svcerr_systemerr( )**
- **svcerr_weakauth( )**

rpc_svc_reg(3N) Library routines for RPC servers. The routines documented on this page include:

- **svc_fds( )**
- **svc_fdset( )**
- **svc_freeargs( )**
- **svc_getargs( )**
- **svc_getcaller( )**
- **svc_getreq( )**
- **svc_getreqset( )**
- **svc_run( )**
- **svc_sendreply( )**

rpc_xdr(3N) XDR library routines for remote procedure calls. The routines documented on this page include:

- **xdr_accepted_reply( )**
- **xdr_authunix_parms( )**
- **xdr_callhdr( )**
- **xdr_callmsg( )**
- **xdr_opaque_auth( )**
- **xdr_rejected_reply( )**
- **xdr_replymsg( )**

secure_rpc(3N) Library routines for secure remote procedure calls. The routines documented on this page include:

authdes_create( )
authdes_getucred( )
get_mayaddress( )
getnetname( )
host2netname( )
key_decryptsession( )
key_encryptsession( )
key_gendes( )
key_setsecret( )
netname2host( )
netname2user( )
user2netname( )

SEE ALSO

portmap(3N), **rpc_clnt_auth**(3N), **rpc_clnt_calls**(3N), **rpc_clnt_create**(3N), **rpc_svc_calls**(3N), **rpc_svc_create**(3N), **rpc_svc_err**(3N), **rpc_svc_reg**(3N), **rpc_xdr**(3N), **secure_rpc**(3N), **xdr**(3N), **publickey**(5), **portmap**(8C), **keyserv**(8C)

Network Programming

NAME

auth_destroy, authnone_create, authunix_create, authunix_create_default – library routines for client side remote procedure call authentication

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

RPC allows various authentication types. Currently, it supports **AUTH_NONE, AUTH_UNIX, AUTH_DES**. For routines relating to the **AUTH_DES** type, see **secure_rpc**(3N).

These routines are called after creating the CLIENT handle. The client's authentication information is passed to the server when the RPC call is made.

Routines

The following routines require that the header **<rpc.h>**. be included. The **AUTH** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/rpc.h>

void auth_destroy(auth)
AUTH *auth;

Destroy the authentication information associated with *auth*. Destruction usually involves deallocation of private data structures. The use of *auth* is undefined after calling **auth_destroy( )**.

AUTH * authnone_create( )

Create and return an RPC authentication handle that passes no usable authentication information with each remote procedure call. This is the default authentication used by RPC.

AUTH * authunix_create(host, uid, gid, grouplen, gidlistp)
char *host;
int uid, gid, grouplen, *gidlistp;

Create and return an RPC authentication handle that contains authentication information. The parameter *host* is the name of the machine on which the information was created; *uid* is the user's user ID; *gid* is the user's current group ID; *grouplen* and *gidlistp* refer to a counted array of groups to which the user belongs. Warning: It is not very difficult to impersonate a user.

AUTH * authunix_create_default( )

Call **authunix_create( )** with the appropriate parameters.

SEE ALSO

rpc(3N), **rpc_clnt_create**(3N), **rpc_clnt_calls**(3N)

NAME

callrpc, clnt_broadcast, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror – library routines for client side calls

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

The **clnt_call( )**, **callrpc( )** and **clnt_broadcast( )** routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.

Routines

The **CLIENT** data structure is defined in the RPC/XDR Library Definition of the *Network Programming*.

#include <rpc/rpc.h>

```
int callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
char *host;
u_long prognum, versnum, procnum;
char *in;
xdrproc_t inproc;
char *out;
xdrproc_t outproc;
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *in* is the address of the procedure's argument, and *out* is the address of where to place the result; *inproc* is an XDR function used to encode the procedure's parameters, and *outproc* is an XDR function used to decode the procedure's results. This routine returns 0 if it succeeds, or the value of **enum clnt_stat** cast to an integer if it fails. Use **clnt_perrno( )** to translate failure statuses into messages.

Warning: Calling remote procedures with this routine uses UDP/IP as the transport; see **clntudp_create( )** on **rpc_clnt_create**(3N) for restrictions. You do not have control of timeouts or authentication using this routine.

```
enum clnt_stat clnt_broadcast(prognum, versnum, procnum, inproc, in, outproc, out, eachresult)
u_long prognum, versnum, procnum;
char *in;
xdrproc_t inproc;
char *out;
xdrproc_t outproc;
bool_t eachresult;
```

Like **callrpc( )**, except the call message is broadcast to all locally connected broadcast nets. Each time the caller receives a response, this routine calls **eachresult( )**, whose form is:

```
int eachresult(out, addr)
char *out;
struct sockaddr_in *addr;
```

where *out* is the same as *out* passed to **clnt_broadcast( )**, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results. If **eachresult( )** returns 0 **clnt_broadcast( )** waits for more replies; otherwise it returns with appropriate status. If **eachresult( )** is NULL, **clnt_broadcast( )** returns without waiting for any replies.

Note: **clnt_broadcast()** uses **AUTH_UNIX** style of authentication.

Warning: Broadcast packets are limited in size to the maximum transfer unit of the data link. For Ethernet, the callers argument size should not exceed 1400 bytes.

enum clnt_stat clnt_call(clnt, procnum, inproc, in, outproc, out, timeout)
CLIENT *clnt;
u_long procnum;
xdrproc_t inproc, outproc;
char *in, *out;
struct timeval timeout;

Call the remote procedure *procnum* associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as **clnt_create()** (see **rpc_clnt_create**(3N). The parameter *in* is the address of the procedure's argument, and *out* is the address of where to place the result; *inproc* is an XDR function used to encode the procedure's parameters in XDR, and *outproc* is used to decode the procedure's results; *timeout* is the time allowed for a response from the server.

bool_t clnt_freeres(clnt, outproc, out)
CLIENT *clnt;
xdrproc_t outproc;
char *out;

Free any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter *out* is the address of the results, and *outproc* is the XDR routine describing the results. This routine returns TRUE if the results were successfully freed, and FALSE otherwise. Note: This is equivalent to doing **xdr_free(outproc, out)** (see **xdr_simple**(3N)).

void clnt_geterr(clnt, errp)
CLIENT *clnt;
struct rpc_err *errp;

Copy the error structure out of the client handle to the structure at address *errp*. *errp* should point to preallocated space.

void clnt_perrno(stat)
enum clnt_stat stat;

Print a message to the standard error corresponding to the condition indicated by *stat*. A NEWLINE is appended at the end of the message. Used after **callrpc()** or **clnt_broadcast()**.

void clnt_perror(clnt, str)
CLIENT *clnt;
char *str;

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A NEWLINE is appended at the end of the message. Used after **clnt_call()**.

char *clnt_sperrno(stat)
enum clnt_stat stat;

Take the same arguments as **clnt_perrno()**, but instead of sending a message to the standard error indicating why an RPC failed, return a pointer to a string which contains the message. **clnt_sperrno()** does not append a NEWLINE at the end of the message.

clnt_sperrno() is used instead of **clnt_perrno()** if the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with **printf**(3V), or if a message format different than that supported by **clnt_perrno()** is to be used.

char *clnt_sperror(clnt, str)
CLIENT *clnt;
char *str;

Like **clnt_perror( )**, except that (like **clnt_sperrno( )**) it returns a string instead of printing to the standard error. Unlike **clnt_perror( )**, it does not append the message with a NEWLINE.

Note: **clnt_sperror( )** returns pointer to a static buffer that is overwritten on each call.

SEE ALSO

printf(3V), **rpc**(3N), **rpc_clnt_auth**(3N), **rpc_clnt_create**(3N), **xdr_simple**(3N)

NAME

clnt_control, clnt_create, clnt_create_vers, clnt_destroy, clnt_pcreateerror, clntraw_create, clnt_spcreateerror, clnttcp_create, clntudp_bufcreate, rpc_createrr – library routines for dealing with creation and manipulation of CLIENT handles

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

The **CLIENT** data structure is defined in the RPC/XDR Library Definition of the *Network Programming*.

#include <rpc/rpc.h>

bool_t clnt_control(clnt, request, info)
CLIENT *clnt;
int request;
char *info;

Change or retrieve various information about a client object. *request* indicates the type of operation, and *info* is a pointer to the information. For both UDP and TCP, the supported values of *request* and their argument types and what they do are:

CLSET_TIMEOUT	struct timeval	set total timeout
CLGET_TIMEOUT	struct timeval	get total timeout
CLGET_FD	int	get associated socket
CLSET_FD_CLOSE	void	close socket on **clnt_destroy()**
CLSET_FD_NCLOSE	void	leave socket open on **clnt_destroy()**

Note: If you set the timeout using **clnt_control()**, the timeout parameter passed to **clnt_call()** (see **rpc_clnt_calls**(3N)) will be ignored in all future calls.

CLGET_SERVER_ADDR	struct sockaddr_in	get server's address

The following operations are valid for UDP only:

CLSET_RETRY_TIMEOUT	struct timeval	set the retry timeout
CLGET_RETRY_TIMEOUT	struct timeval	get the retry timeout

The retry timeout is the time that UDP RPC waits for the server to reply before retransmitting the request.

This routine returns TRUE on success, and FALSE on failure.

CLIENT * clnt_create(host, prognum, versnum, protocol)
char *host;
u_long prognum, versnum;
char *protocol;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *protocol* indicates which kind of transport protocol to use. The currently supported values for this field are "udp" and "tcp". Default timeouts are set, but they can be modified using **clnt_control()**. If successful it returns a client handle, otherwise it returns NULL.

Warning: Using UDP has its shortcomings. Since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take arguments or return results larger than 8 Kbytes. Use TCP instead.

Note: If the requested version number *versnum* is not registered with the **portmap**(8C) service on *host*, but at least a version number for the given program number is registered, **clnt_create**( ) returns a handle. The version mismatch will be discovered by a **clnt_call**( ) later (see **rpc_clnt_calls**(3N)).

CLIENT * **clnt_create_vers(host, prognum, vers_outp, vers_low, vers_high, protocol)**
char *host;
u_long prognum;
u_long *vers_outp;
u_long vers_low, vers_high;
char *protocol;

This is a generic client creation routine which also checks for the version available. *host* identifies the name of the remote host where the server is located. *protocol* indicates which kind of transport protocol to use. The currently supported values for this field are "udp" and "tcp". If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. Default timeouts are set, but can be modified using **clnt_control**( ).

Note: **clnt_create**( ) returns a valid client handle even if the particular version number supplied to **clnt_create**( ) is not registered with the portmap service. This mismatch will be discovered by a **clnt_call**( ) later (see **rpc_clnt_calls**(3N)). However, **clnt_create_vers**( ) does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

void clnt_destroy(clnt)
CLIENT ***clnt;**

Destroy the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling **clnt_destroy**( ). If the RPC library opened the associated socket, or CLSET_FD_CLOSE was set using **clnt_control**( ). **clnt_destroy**( ) closes the socket.

void clnt_pcreateerror(str)
char *str;

Print a message to the standard error indicating why a client handle could not be created. The message is prepended with string *s* and a colon. Used when routines such as **clnt_create**( ), **clntraw_create**( ), **clnttcp_create**( ), or **clntudp_create**( ) fails.

CLIENT * **clntraw_create(prognum, versnum)**
u_long prognum, versnum;

Create an RPC client for the remote program *prognum*, version *versnum*. The transport used to pass messages to the service is actually a buffer within the process's address space, so the corresponding RPC server should live in the same address space; also see **svcraw_create**( ) (see **rpc_svc_create**(3N)). This allows simulation of RPC and getting RPC overheads, such as round trip times, without any kernel interference. If successful it returns a client handle, otherwise it returns NULL.

char * clnt_spcreateerror(str)
char *str;

Like **clnt_pcreateerror()**, except that it returns a string instead of printing to the standard error. It, however, does not append the message with a NEWLINE.

Note: **clnt_spcreateerror()** returns a pointer to a static buffer that is overwritten on each call.

CLIENT * clnttcp_create(addr, prognum, versnum, sockp, sendsz, recvsz)
struct sockaddr_in *addr;
u_long prognum, versnum;
int *sockp;
u_int sendsz, recvsz;

Create a client handle for the remote program *prognum*, version *versnum*; the client uses TCP/IP as a transport. The remote program is located at Internet address *addr*. If **addr->sin_port** is zero, it is set to the port on which the remote program is listening (the remote **portmap** service is consulted for this information). The parameter *sockp* is a pointer to a socket; if it is **RPC_ANYSOCK**, then a new socket is opened and *sockp* is updated. Since TCP-based RPC uses buffered I/O, the user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of zero choose defaults. If successful it returns a client handle, otherwise it returns NULL.

Warning: If **addr->sin_port** is zero and the requested version number *versnum* is not registered with the remote portmap service, it returns a handle if at least a version number for the given program number is registered. The version mismatch will be discovered by a **clnt_call()** later (see **rpc_clnt_calls**(3N)).

CLIENT * clntudp_bufcreate(addr, prognum, versnum, wait, sockp, sendsz, recvsz)
struct sockaddr_in *addr;
u_long prognum, versnum;
struct timeval wait;
int *sockp;
u_int sendsz;
u_int recvsz;

Create a client handle for the remote program *prognum*, on *versnum*; the client uses UDP/IP as the transport. The remote program is located at the Internet address *addr*. If **addr->sin_port** is zero, it is set to port on which the remote program is listening on (the remote **portmap** service is consulted for this information). The parameter *sockp* is a pointer to a socket; if it is **RPC_ANYSOCK**, then a new socket is opened and **sockp** is updated. The UDP transport resends the call message in intervals of *wait* time until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call()** (see **rpc_clnt_calls**(3N)). If successful it returns a client handle, otherwise it returns NULL.

The user can specify the maximum packet size for sending and receiving by using *sendsz* and *recvsz* arguments for UDP-based RPC messages.

Warning: If **addr->sin_port** is zero and the requested version number *versnum* is not registered with the remote portmap service, it returns a handle if at least a version number for the given program number is registered. The version mismatch is discovered by a **clnt_call()** later (see **rpc_clnt_calls**(3N)).

CLIENT * clntudp_create(addr, prognum, versnum, wait, sockp)
struct sockaddr_in *addr;
u_long prognum, versnum;
struct timeval wait;
int *sockp;

Create a client handle for the remote program *prognum*, version *versnum*; the client uses UDP/IP as the transport. The remote program is located at the Internet address *addr*. If **addr->sin_port** is zero, then it is set to actual port that the remote program is listening on (the remote **portmap** service is consulted for this information). The parameter *sockp* is a pointer to a socket; if it is **RPC_ANYSOCK**, a new socket is opened and *sockp* is updated. The UDP transport resends the call message in intervals of *wait* time until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call()** (see **rpc_clnt_calls**(3N)). If successful it returns a client handle, otherwise it returns NULL.

Warning: Since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take arguments or results larger than 8 Kbytes. TCP should be used instead.

Warning: If **addr->sin_port** is zero and the requested version number *versnum* is not registered with the remote portmap service, it returns a handle if any version number for the given program number is registered. The version mismatch is be discovered by a **clnt_call()** later (see **rpc_clnt_calls**(3N)).

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine **clnt_pcreateerror()** to print the reason for the failure.

SEE ALSO

portmap(3N), **rpc**(3N), **rpc_clnt_auth**(3N), **rpc_clnt_calls**(3N), **rpc_svc_create**(3N)

NAME

registerrpc, svc_register, svc_unregister, xprt_register, xprt_unregister – library routines for registerring servers

DESCRIPTION

These routines are a part of the RPC library which allows the RPC servers to register themselves with **portmap**(8C), and it associates the given program and version number with the dispatch function.

Routines

The **SVCXPRT** data structure is defined in the RPC/XDR Library Definition of the *Network Programming*.

#include <rpc/rpc.h>

```
int registerrpc(prognum, versnum, procnum, procname, inproc, outproc)
u_long prognum, versnum, procnum;
char *(*procname) () ;
xdrproc_t inproc, outproc;
```

Register procedure *procname* with the RPC service package. If a request arrives for program *prognum*, version *versnum*, and procedure *procnum*, *procname* is called with a pointer to its parameter; *progname* must be a procedure that returns a pointer to its static result; *inproc* is used to decode the parameters while *outproc* is used to encode the results. This routine returns 0 if the registration succeeded, −1 otherwise.

Warning: Remote procedures registered in this form are accessed using the UDP/IP transport; see **svcudp_create( )** on **rpc_svc_create**(3N) for restrictions. This routine should not be used more than once for the same program and version number.

```
bool_t svc_register(xprt, prognum, versnum, dispatch, protocol)
SVCXPRT *xprt;
u_long prognum, versnum;
void (*dispatch) ();
u_long protocol;
```

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *protocol* is zero, the service is not registered with the **portmap** service. If *protocol* is non-zero, a mapping of the triple [*prognum, versnum, protocol*] to **xprt–>xp_port** is established with the local **portmap** service (generally *protocol* is zero, **IPPROTO_UDP** or **IPPROTO_TCP**). The procedure *dispatch* has the following form:

```
dispatch(request, xprt)
struct svc_req *request;
SVCXPRT *xprt;
```

The **svc_register( )** routine returns TRUE if it succeeds, and FALSE otherwise.

```
void svc_unregister(prognum, versnum)
u_long prognum, versnum;
```

Remove all mapping of the pair [*prognum*,*versnum*] to dispatch routines, and of the triple [*prognum*,*versnum*,*] to port number.

```
void xprt_register(xprt)
SVCXPRT *xprt;
```

After RPC service transport handles are created, they should register themselves with the RPC service package. This routine modifies the global variable **svc_fds**. Service implementors usually do not need this routine.

void xprt_unregister(xprt)
SVCXPRT *xprt;

Before an RPC service transport handle is destroyed, it should unregister itself with the RPC service package. This routine modifies the global variable **svc_fds**. Service implementors usually do not need this routine directly.

SEE ALSO

portmap(3N), **rpc**(3N), **rpc_svc_err**(3N), **rpc_svc_create**(3N), **rpc_svc_reg**(3N), **portmap**(8C)

NAME

svc_destroy, svcfd_create, svcraw_create, svctcp_create, svcudp_bufcreate – library routines for dealing with the creation of server handles

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

The **SVCXPRT** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/rpc.h>

void svc_destroy(xprt)
SVCXPRT *xprt;

> Destroy the RPC service transport handle, *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT * svcfd_create(fd, sendsz, recvsz)
int fd;
u_int sendsz;
u_int recvsz;

> Create a service on top of any open and bound descriptor and return the handle to it. Typically, this descriptor is a connected socket for a stream protocol such as TCP. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are zero, a reasonable default is chosen. It returns NULL if it fails.

SVCXPRT * svcraw_create()

> This routine creates a RPC service transport, to which it returns a pointer. The transport is a buffer within the process's address space, so the corresponding RPC client must live in the same address space; see **clntraw_create()** on **rpc_clnt_create**(3N). This routine allows simulation of RPC and getting RPC overheads (such as round trip times), without any kernel interference. This routine returns NULL if it fails.

SVCXPRT * svctcp_create(sock, sendsz, recvsz)
int sock;
u_int sendsz, recvsz;

> This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket *sock*. If sock is **RPC_ANYSOCK**, then a new socket is created. If the socket is not bound to a local TCP port, then this routine binds it to an arbitrary port. Upon completion, **xprt–>xp_sock** is the transport's socket descriptor, and **xprt–>xp_port** is the port number on which it is listening. This routine returns NULL if it fails. Since TCP-based RPC uses buffered I/O, users may specify the size of buffers with *sendsz* and *recvsz*; values of zero choose defaults.

SVCXPRT * svcudp_bufcreate(sock, sendsz, recvsz)
int sock;
u_int sendsz, recvsz;

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket *sock*. If sock is **RPC_ANYSOCK** , then a new socket is created. If the socket is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, **xprt->xp_sock** is the service's socket descriptor, and **xprt->xp_port** is the service's port number. This routine returns NULL if it fails.

The user specifies the maximum packet size for sending and receiving UDP-based RPC messages by using the *sendsz* and *recvsz* parameters.

SEE ALSO

rpc(3N), **rpc_clnt_create**(3N), **rpc_svc_calls**(3N), **rpc_svc_err**(3N), **rpc_svc_reg**(3N), **portmap**(8C)

NAME

svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprog, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

These routines can be called by the server side dispatch function if there is any error in the transaction with the client.

Routines

The **SVCXPRT** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/rpc.h>

void svcerr_auth(xprt, why)
SVCXPRT *xprt;
enum auth_stat why;

Called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.

void svcerr_decode(xprt)
SVCXPRT *xprt;

Called by a service dispatch routine that cannot successfully decode the remote parameters. See **svc_getargs( )** in **rpc_svc_reg**(3N).

void svcerr_noproc(xprt)
SVCXPRT *xprt;

Called by a service dispatch routine that does not implement the procedure number that the caller requests.

void svcerr_noprog(xprt)
SVCXPRT *xprt;

Called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.

void svcerr_progvers(xprt)
SVCXPRT *xprt;

Called when the desired version of a program is not registered with the RPC package. Service implementors usually do not need this routine.

void svcerr_systemerr(xprt)
SVCXPRT *xprt;

Called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

void svcerr_weakauth(xprt)
SVCXPRT *xprt;

Called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient authentication parameters. The routine calls **svcerr_auth(xprt, AUTH_TOOWEAK)**.

SEE ALSO

rpc(3N), **rpc_svc_calls**(3N), **rpc_svc_create**(3N), **rpc_svc_reg**(3N)

NAME

svc_fds, svc_fdset, svc_freeargs, svc_getargs, svc_getcaller, svc_getreq, svc_getreqset, svc_getcaller, svc_run, svc_sendreply – library routines for RPC servers

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as **svc_run()**) are called when the server is initiated.

Routines

The **SVCXPRT** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/rpc.h>

int svc_fds;

Similar to **svc_fdset**, but limited to 32 descriptors. This interface is obsoleted by **svc_fdset**.

fd_set svc_fdset;

A global variable reflecting the RPC server's read file descriptor bit mask; it is suitable as a parameter to the **select()** system call. This is only of interest if a service implementor does not call **svc_run()**, but rather does their own asynchronous event processing. This variable is read-only (do not pass its address to **select()**!), yet it may change after calls to **svc_getreqset()** or any creation routines.

bool_t svc_freeargs(xprt, inproc, in)
SVCXPRT *xprt;
xdrproc_t inproc;
char *in;

Free any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using **svc_getargs()**. This routine returns TRUE if the results were successfully freed, and FALSE otherwise.

bool_t svc_getargs(xprt, inproc, in)
SVCXPRT *xprt;
xdrproc_t inproc;
char *in;

Decode the arguments of an RPC request associated with the RPC service transport handle, *xprt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns TRUE if decoding succeeds, and FALSE otherwise.

struct sockaddr_in * svc_getcaller(xprt)
SVCXPRT *xprt;

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle, *xprt*.

void svc_getreq(rdfds)
int rdfds;

Similar to **svc_getreqset()**, but limited to 32 descriptors. This interface is obsoleted by **svc_getreqset()**.

void svc_getreqset(rdfdsp)
fd_set *rdfdsp;

This routine is only of interest if a service implementor does not use **svc_run()**, but instead implements custom asynchronous event processing. It is called when the **select()** system call has determined that an RPC request has arrived on some RPC **socket(s)** ; *rdfdsp* is the resultant read file descriptor bit mask. The routine returns when all sockets associated with the value of *rdfdsp* have been serviced.

void svc_run()

Normally, this routine only returns in the case of some errors. It waits for RPC requests to arrive, and calls the appropriate service procedure using **svc_getreq()** when one arrives. This procedure is usually waiting for a **select()** system call to return.

bool_t svc_sendreply(xprt, outproc, out)
SVCXPRT *xprt;
xdrproc_t outproc;
char *out;

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns TRUE if it succeeds, FALSE otherwise.

SEE ALSO

select(2), **rpc**(3N), **rpc_svc_calls**(3N), **rpc_svc_create**(3N), **rpc_svc_err**(3N)

NAME

xdr_accepted_reply, xdr_authunix_parms, xdr_callhdr, xdr_callmsg, xdr_opaque_auth, xdr_rejected_reply, xdr_replymsg – XDR library routines for remote procedure calls

DESCRIPTION

These routines are used for describing the RPC messages in XDR language. They should normally be used by those who do not want to use the RPC package.

Routines

The **XDR** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming.*

#include <rpc/rpc.h>

bool_t xdr_accepted_reply(xdrs, arp)
XDR *xdrs;
struct accepted_reply *arp;

Used for encoding RPC reply messages. It encodes the status of the RPC call in the XDR language format and in the case of success, it encodes the call results as well. This routine is useful for users who wish to generate RPC-style messages without using the RPC package. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_authunix_parms(xdrs, aup)
XDR *xdrs;
struct authunix_parms *aup;

Used for describing UNIX credentials. It encludes machine name, user ID, group ID list, etc. This routine is useful for users who wish to generate these credentials without using the RPC authentication package. This routine returns TRUE if it succeeds, FALSE otherwise.

void xdr_callhdr(xdrs, chdrp)
XDR *xdrs;
struct rpc_msg *chdrp;

Used for describing RPC call header messages. It encodes the static part of the call message header in the XDR language format. It includes information such as transaction ID, RPC version number, program number, and version number. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

bool_t xdr_callmsg(xdrs, cmsgp)
XDR *xdrs;
struct rpc_msg *cmsgp;

Used for describing RPC call messages. It includes all the RPC call information such as transaction ID, RPC version number, program number, version number, authentication information, etc. This routine is useful for users who wish to generate RPC-style messages without using the RPC package. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_opaque_auth(xdrs, ap)
XDR *xdrs;
struct opaque_auth *ap;

Used for describing RPC authentication information messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_rejected_reply(xdrs, rrp)
XDR *xdrs;
struct rejected_reply *rrp;

Used for describing RPC reply messages. It encodes the rejected RPC message in the XDR language format. The message is rejected either because of version number mismatch or because of authentication errors. This routine is useful for users who wish to generate RPC-style messages without using the RPC package. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_replymsg(xdrs, rmsgp)
XDR *xdrs;
struct rpc_msg *rmsgp;

Used for describing RPC reply messages. It encodes the RPC reply message in the XDR language format. This reply could be an acceptance, rejection, or NULL. This routine is useful for users who wish to generate RPC style messages without using the RPC package. This routine returns TRUE if it succeeds, FALSE otherwise.

SEE ALSO

rpc(3N)

NAME

rtime – get remote time

SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>

int rtime(addrp, timep, timeout)
struct sockaddr_in *addrp;
struct timeval *timep;
struct timeval *timeout;
```

DESCRIPTION

rtime( ) consults the Internet Time Server at the address pointed to by *addrp* and returns the remote time in the **timeval** struct pointed to by *timep*. Normally, the UDP protocol is used when consulting the Time Server. The *timeout* parameter specifies how long the routine should wait before giving up when waiting for a reply. If *timeout* is specified as NULL, however, the routine will instead use TCP and block until a reply is received from the time server.

The routine returns 0 if it is successful. Otherwise, it returns −1 and **errno** is set to reflect the cause of the error.

NAME

scandir, alphasort – scan a directory

SYNOPSIS

```
#include &lt;sys/types.h&gt;
#include &lt;sys/dir.h&gt;

scandir(dirname, &namelist, select, compar)
char *dirname;
struct direct **namelist;
int (*select)( );
int (*compar)( );

alphasort(d1, d2)
struct direct **d1, **d2;
```

DESCRIPTION

scandir( ) reads the directory **dirname** and builds an array of pointers to directory entries using **malloc**(3V). The second parameter is a pointer to an array of structure pointers. The third parameter is a pointer to a routine which is called with a pointer to a directory entry and should return a non zero value if the directory entry should be included in the array. If this pointer is NULL, then all the directory entries will be included. The last argument is a pointer to a routine which is passed to **qsort**(3) to sort the completed array. If this pointer is NULL, the array is not sorted. **alphasort**( ) is a routine which will sort the array alphabetically.

scandir( ) returns the number of entries in the array and a pointer to the array through the parameter *namelist.*

SEE ALSO

directory(3V), **malloc**(3V), **qsort**(3)

DIAGNOSTICS

Returns –1 if the directory cannot be opened for reading or if **malloc**(3V) cannot allocate enough memory to hold all the data structures.

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

#include <stdio.h>

int scanf(format [, pointer . . .])
char *format;

int fscanf(stream, format [, pointer . . .])
FILE *stream;
char *format;

int sscanf(s, format [, pointer . . .])
char *s, *format;

SYSTEM V SYNOPSIS

The following are provided for XPG2 compatibility:

#define nl_scanfscanf
#define nl_fscanf fscanf
#define nl_sscanf sscanf

DESCRIPTION

scanf( ) reads from the standard input stream **stdin. fscanf( )** reads from the named input stream. **sscanf( )** reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored. The results are undefined in there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

- White-space characters (SPACE, TAB, or NEWLINE) which, except in two cases described below, cause input to be read up to the next non-white-space character.
- An ordinary character (not '%'), which must match the next character of the input stream.
- Conversion specifications, consisting of the character '%' or the character sequence *%digit*$, an optional assignment suppressing character '*', an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

Conversion specifications are introduced by the character % or the character sequence *%digit*$. A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by '*'. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except ''['' and ''c'', white space leading an input field is ignored.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion characters are legal:

- **%** A single % is expected in the input at this point; no assignment is done.
- **d** A decimal integer is expected; the corresponding argument should be an integer pointer.
- **u** An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
- **o** An octal integer is expected; the corresponding argument should be an integer pointer.
- **x** A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

i An integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.

n Stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.

e,f,g A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is as described for **string_to_decimal**(3), with *fortran_conventions* zero.

s A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white space character.

c A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

[Indicates string data; the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating **\0**, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, **x**, and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

Avoid this common error: because **printf**(3V) does not require that the lengths of conversion descriptors and actual parameters match, coders sometimes are careless with the **scanf**( ) functions. But converting %f to &double or %lf to &float *does not work*; the results are quite incorrect.

scanf( ) conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf( ) returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. The constant EOF is returned upon end of input. Note: this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

If the input ends before the first conflict or conversion, EOF is returned. If the input ends after the first conflict or conversion, the number of successfully matched items is returned.

Conversions can be applied to the *n*th argument in the argument list, rather than the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %digit$, where *digit* is a decimal integer *n* in the range [1,9], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages.

The format string can contain either form of a conversion specification, that is % or %*digit*$, although the two forms cannot be mixed within a single format string.

All forms of the **scanf()** functions allow for the detection of a language dependent radix character in the input string. The radix character is defined by the program's locale (category **LC_NUMERIC**). In the "C" locale, or in a locale where the radix character is not defined, the radix character defaults to '.'.

SYSTEM V DESCRIPTION

FORMFEED is allowed as a white space character in control strings.

XPG2 requires that **nl_scanf**, **nl_fscanf** and **nl_sscanf** be defined as **scanf**, **fscanf** and **sscanf**, respectively for backward compatibility.

RETURN VALUES

If any items are converted, **scanf()**, **fscanf()** and **sscanf()** return the number of items converted successfully. This number may smaller than the number of items requested. If no items are converted, these functions return 0. **scanf()**, **fscanf()** and **sscanf()** return EOF on end of input.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E–1 thompson
```

will assign to *n* the value 3, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i, j; float x; char name[50];
(void) scanf("%i%2d%f%*d %[0–9]", &j, &i, &x, name);
```

with input:

```
011 56789 0123 56a72
```

will assign **9** to *j*, **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to **getchar()** (see **getc**(3V)) will return **a**. Or:

```
int i, j, s, e; char name[50];
(void) scanf("%i %i %n%s%n", &i, &j, &s, name, &e);
```

with input:

```
0x11 0xy johnson
```

will assign **17** to *i*, **0** to *j*, **6** to *s*, will place the string **xy\0** in *name*, and will assign **8** to *e*. Thus, the length of *name* is *e* – *s* = 2. The next call to **getchar()** (see **getc**(3V)) will return a SPACE.

SEE ALSO

getc(3V), **printf**(3V), **setlocale**(3V), **stdio**(3V), **string_to_decimal**(3), **strtol**(3)

WARNINGS

Trailing white space (including a NEWLINE) is left unread unless matched in the control string.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

authdes_create, authdes_getucred, get_myaddress, getnetname, host2netname, key_decryptsession, key_encryptsession, key_gendes, key_setsecret, netname2host, netname2user, user2netname – library routines for secure remote procedure calls

DESCRIPTION

RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

RPC allows various authentication flavors The **authdes_getucred()** and **authdes_create()** routines implement the DES authentication flavor. See **rpc_clnt_auth**(3N) for routines relating to the **AUTH_NONE** and **AUTH_UNIX** authentication types.

> Note: Both the client and server should have their keys in the **publickey**(5) database. Also, the keyserver daemon **keyserv**(8C) must be running on both the client and server hosts for the DES authentication system to work.

Routines

```
#include <rpc/rpc.h>
```

```
AUTH * authdes_create(netname, window, syncaddr, deskeyp)
char *netname;
unsigned window;
struct sockaddr_in *syncaddr;
des_block *deskeyp;
```

> **authdes_create()** is an interface to the RPC secure authentication system, known as DES authentication.
>
> Used on the client side, **authdes_create()** returns an authentication handle that enables the use of the secure authentication system. The first parameter *netname* is the network name of the owner of the server process. This field usually represents a *host* derived from the utility routine **host2netname()**, but could also represent a user name using **user2netname()**. The second field is window on the validity of the client credential, given in seconds. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter *syncaddr* is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server's clock, and will not attempt to synchronize with the server. If an address is supplied then the system will use it for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter *deskeyp* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If *deskeyp* is supplied then it is used instead.

```
int authdes_getucred(adc, uidp, gidp, gidlenp, gidlistp)
struct authdes_cred *adc;
short *uidp;
short *gidp;
short *gidlenp;
int *gidlistp;
```

> **authdes_getucred()**, is a DES authentication routine used by the server for converting a DES credential, which is operating system independent, into a UNIX credential. *uidp* points to the user ID of the user associated with *adc*; *gidp* refers to the user's current group ID; *gidlistp* refers to an array of groups to which the user belongs and *gidlenp* has the count of the entries in this array.

This routine differs from the utility routine **netname2user()** in that **authdes_getucred()** pulls its information from a cache, and does not have to do a NIS name service lookup every time it is called to get its information. Returns 1 if it succeeds and 0 if it fails.

void get_myaddress(addr)
struct sockaddr_in *addr;

Return the machine's IP address in *addr*. The port number is always set to **htons(PMAPPORT)**.

int getnetname(netname)
char netname[MAXNETNAMELEN];

Return the unique, operating-system independent netname of the caller in the fixed-length array *netname*. Returns 1 if it succeeds and 0 if it fails.

int host2netname(netname, host, domain)
char netname[MAXNETNAMELEN];
char *host;
char *domain;

Convert from a domain-specific hostname to an operating-system independent netname. This routine is normally used to get the netname of the server, which is then used to get an authentication handle by calling **authdes_create()**. This routine should be used if the owner of the server process is the machine that is, the user with effective user ID zero. Returns 1 if it succeeds and 0 if it fails. This routine is the inverse of **netname2host()**.

int key_decryptsession(netname, deskeyp)
char *netname;
des_block *deskeyp;

An interface routine to the keyserver daemon, which is associated with RPC's secure authentication system (DES authentication). User programs rarely need to call it, or its associated routines **key_encryptsession()**, **key_gendes()** and **key_setsecret()**. System commands such as **login** and the RPC library are the main clients of these four routines.

key_decryptsession() takes the netname of a server and a DES key, and decrypts the key by using the public key of the server and the secret key associated with the effective user ID of the calling process. Returns 0 if it succeeds and −1 if it fails. This routine is the inverse of **key_encryptsession()**.

int key_encryptsession(netname, deskeyp)
char *netname;
des_block *deskeyp;

A keyserver interface routine. It takes the netname of the server and a des key, and encrypts it using the public key of the server and the secret key associated with the effective user ID of the calling process. Returns 0 if it succeeds and −1 if it fails. This routine is the inverse of **key_decryptsession()**.

int key_gendes(deskeyp)
des_block *deskeyp;

A keyserver interface routine. It is used to ask the keyserver for a secure conversation key. Choosing one at "random" is usually not good enough, because the common ways of choosing random numbers, such as using the current time, are very easy to guess. Returns 0 if it succeeds and −1 if it fails.

int key_setsecret(keyp)
char *keyp;

A keyserver interface routine. It is used to set the secret key for the effective user ID of the calling process. Returns 0 if it succeeds and −1 if it fails.

int netname2host(netname, host, hostlen)
char *netname;
char *host;
int hostlen;

Convert an operating-system independent netname to a domain-specific hostname. *hostlen* specifies the size of the array pointed to by *host.* It returns 1 if it succeeds and 0 if it fails. This routine is the inverse of **host2netname( )**.

int netname2user(netname, uidp, gidp, gidlenp, gidlistp)
char *name;
int *uidp;
int *gidp;
int *gidlenp;
int *gidlistp;

Convert an operating-system independent netname to a domain-specific user ID. *uidp* points to the user ID of the user; *gidp* refers to the user's current group ID; *gidlistp* refers to an array of groups to which the user belongs and *gidlenp* has the count of the entries in this array. It returns 1 if it succeeds and 0 if it fails. This routine is the inverse of **user2netname( )**.

int user2netname(netname, uid, domain)
char name[MAXNETNAMELEN];
int uid;
char *domain;

Convert a domain-specific username to an operating-system independent netname. *uid* is the user ID of the owner of the server process. This routine is normally used to get the netname of the server, which is then used to get an authentication handle by calling **authdes_create( )**. Returns 1 if it succeeds and 0 if it fails. This routine is the inverse of **netname2user( )**.

SEE ALSO

login(1), **chkey**(1), **rpc**(3N), **rpc_clnt_auth**(3N), **publickey**(5), **keyserv**(8C), **newkey**(8)

NAME

setbuf, setbuffer, setlinebuf, setvbuf – assign buffering to a stream

SYNOPSIS

#include <stdio.h>

void setbuf(stream, buf)
FILE *stream;
char *buf;

void setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

int setlinebuf(stream) FILE *stream;

int setvbuf(stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from **stdin**. **fflush**() (see **fclose**(3V)) may be used to force the block out early. A buffer is obtained from **malloc**(3V) upon the first **getc**(3V) or **putc**(3S) on the file. By default, output to a terminal is line buffered, except for output to the standard stream **stderr** which is unbuffered. All other input/output is fully buffered.

setbuf() can be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered. A manifest constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

char buf[BUFSIZ];

setbuffer(), an alternate form of **setbuf**(), can be used after a stream has been opened but before it is read or written. It uses the character array *buf* whose size is determined by the *size* argument instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

setvbuf() can be used after a stream has been opened but before it is read or written. *type* determines how stream will be buffered. Legal values for *type* (defined in **<stdio.h>**) are:

_IOFBF fully buffers the input/output.

_IOLBF line buffers the output; the buffer will be flushed when a NEWLINE is written, the buffer is full, or input is requested.

_IONBF completely unbuffers the input/output.

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used.

setlinebuf() is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike **setbuf**(), **setbuffer**(), and **setvbuf**(), it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using **freopen()** (see **fopen**(3V)). A file can be changed from block buffered or line buffered to unbuffered by using **freopen()** followed by **setbuf()** with a buffer argument of NULL.

SYSTEM V DESCRIPTION

If *buf* is not NULL and *stream* refers to a terminal device, **setbuf()** sets *stream* for line buffered input/output.

RETURN VALUES

setlinebuf() returns no useful value.

setvbuf() returns 0 on success. If an illegal value for *type* or *size* is provided, **setvbuf()** returns a non-zero value. **setvbuf()**

SEE ALSO

fclose(3V), **fopen**(3V), **fread**(3S), **getc**(3V), **malloc**(3V), **printf**(3V), **putc**(3S), **puts**(3S)

NOTES

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

NAME

setjmp, longjmp, sigsetjmp, siglongjmp – non-local goto

SYNOPSIS

#include <setjmp.h>

int setjmp(env)
jmp_buf env;

void longjmp(env, val)
jmp_buf env;
int val;

int _setjmp(env)
jmp_buf env;

void _longjmp(env, val)
jmp_buf env;
int val;

int sigsetjmp(env, savemask)
sigjmp_buf env;
int savemask;

void siglongjmp(env, val)
sigjmp_buf env;
int val;

DESCRIPTION

setjmp( ) and **longjmp**( ) are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

The macro **setjmp**( ) saves its stack environment in *env* for later use by **longjmp**( ). A normal call to **setjmp**( ) returns zero. **setjmp**( ) also saves the register environment. If a **longjmp**( ) call will be made, the routine which called **setjmp**( ) should not return until after the **longjmp**( ) has returned control (see below).

longjmp( ) restores the environment saved by the last call of **setjmp**, and then returns in such a way that execution continues as if the call of **setjmp**( ) had just returned the value *val* to the function that invoked **setjmp**( ); however, if *val* were zero, execution would continue as if the call of **setjmp**( ) had returned one. This ensures that a "return" from **setjmp**( ) caused by a call to **longjmp**( ) can be distinguished from a regular return from **setjmp**( ). The calling function must not itself have returned in the interim, otherwise **longjmp**( ) will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time **longjmp**( ) was called. The CPU and floating-point data registers are restored to the values they had at the time that **setjmp**( ) was called. But, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp**( ). This is especially a problem for programmers trying to write machine-independent C routines.

setjmp( ) and **longjmp**( ) save and restore the signal mask (see **sigsetmask**(2)), while **_setjmp**( ) and **_longjmp**( ) manipulate only the C stack and registers. If the *savemask* flag to **sigsetjmp**( ) is non-zero, the signal mask is saved, and a subsequent **siglongjmp**( ) using the same *env* will restore the signal mask. If the *savemask* flag is zero, the signal mask is not saved, and a subsequent **siglongjmp**( ) using the same *env* will not restore the signal mask. In all other ways, **_setjmp**( ) and **sigsetjmp**( ) function in the same way that **setjmp**( ) does, and **_longjmp**( ) and **siglongjmp**( ) function in the same way that **longjmp**( ) does.

None of these functions save or restore any floating-point status or control registers, in particular the MC68881 **fpsr**, **fpcr**, or **fpiar**, the Sun-3 FPA **fpamode** or **fpastatus**, and the Sun-4 **%fsr**. See **ieee_flags**(3M) to save and restore floating-point status or control information.

SYSTEM V DESCRIPTION

setjmp() and **longjmp()** manipulate only the C stack and registers; they do not save or restore the signal mask. **_setjmp()** behaves identically to **setjmp()**, and **_longjmp()** behaves identically to **longjmp()**.

EXAMPLE

The following code fragment indicates the flow of control of the **setjmp()** and **longjmp()** combination:

```
function declaration
...
        jmp_buf         my_environment;
        ...
        if (setjmp(my_environment)) {
                /* register variables have unpredictable values */
                code after the return from longjmp
                ...
        } else {
                /* do not modify register vars in this leg of code */
                this is the return from setjmp
                ...
        }
```

SEE ALSO

cc(1V), **sigsetmask**(2), **sigvec**(2), **ieee_flags**(3M), **signal**(3V), **setjmp**(3V)

BUGS

setjmp() does not save the current notion of whether the process is executing on the signal stack. The result is that a **longjmp()** to some place on the signal stack leaves the signal stack state incorrect.

On Sun-2 and Sun-3 systems **setjmp()** also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that **setjmp()** was called. All memory-bound data have values as of the time **longjmp()** was called. However, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp()**. When using compiler options that specify automatic register allocation (see **cc**(1V)), the compiler will not attempt to assign variables to registers in routines that call **setjmp()**.

NAME

setlocale, nl_init – set international environment

SYNOPSIS

#include <locale.h>

char *setlocale(category, locale)
int category;
char *locale;

int nl_init(lang)
char *lang;

DESCRIPTION

setlocale() selects the appropriate piece of the program's locale as specified by *category*, and may be used to change or query the program's international environment. The entire locale may be changed by calling **setlocale()** with *category* set to **LC_ALL**. The other possible values for *category* query or change only a part of the program's complete international locale:

LC_CTYPE
: Affects the behavior of the character classification and conversion functions. See **ctype**(3V), and **mblen**(3).

LC_COLLATE
: Affects the behavior of the string collation functions **strcoll (3)** and **strxfrm**(3V).

LC_TIME
: Affects the behavior of the time conversion functions. See **printf**(3V), **scanf**(3V), **strtod**(3), and **ctime**(3V) for **strftime()**, **strptime()**, and **ctime()**.

LC_NUMERIC
: Affects the radix character for the formatted input/output functions and the string conversion functions, **gcvt**(3V), **printf**(3V), **strtod**(3), **gconvert()**, **sgconvert()** (see **econvert**(3)), **file_to_decimal()**, and **func_to_decimal()** (see **string_to_decimal**(3)). Also affects the non-monetary formatting information returned by the **localeconv()** function.

LC_MONETARY
: Affects the monetary formatting information returned by the **localeconv()** function.

LC_MESSAGES
: Affects the behavior of functions that present messages, namely **gettext()**, and **textdomain()**.

The *locale* argument is a pointer to a character string containing the required setting of *category*. The following preset values of *locale* are defined for all settings of *category*:

"C" Specifies the minimal environment for C translation. If **setlocale()** is not invoked, the **"C"** locale is the default. Operational behavior within the **"C"** locale is defined separately for each interface function.

At program startup, the equivalent of:

"" In this case, **setlocale()** will first check the value of the corresponding environment variable (for example, **LC_CTYPE** for the **LC_CTYPE** category) and if valid (that is, points to the name of a valid locale), **setlocale()** sets the specified category of the international environment to that value and returns the string corresponding to the locale set (that is, the value of the environment variable, not ""). If the value is invalid, **setlocale()** returns a NULL pointer and the international environment is not changed by this call.

If the environment variable corresponding to the specified category is not set or is set to the empty string, **setlocale()** will examine the **LANG** environment variable. If both the **LANG** environment variable, and the environment variable corresponding to the specified category are not set or are set to the empty string, then the **LC_default** environment variable is examined. If this contains a valid setting, then the category is set to the value of **LC_default**. If the **LANG** environment variable is set and valid this will set the category to the corresponding value of **LANG**. If **LC_default** is not set, then **setlocale**() returns that category to the default **"C"** locale.

To set all categories in the international environment, **setlocale**() is invoked in the following manner:

setlocale (LC_ALL, "");

To satisfy this request, **setlocale**() first checks all the relevant environment variables **LC_CTYPE, LC_COLLATE, LC_TIME, LC_NUMERIC, LC_MONETARY, LC_MESSAGES.** If any one of these relevant environment variables is invalid, this call to **setlocale**() will return a NULL pointer, and the international environment will not be changed. If all the relevant environment variables are valid, **setlocale**() sets the international environment to reflect the values of the environment variables. The categories are set in the following order:

LC_CTYPE
LC_COLLATE
LC_TIME
LC_NUMERIC
LC_MONETARY
LC_MESSAGES

Using this scheme, the categories corresponding to the environment variables will override the value of the **LANG** and **LC_default** environment variables for a particular category.

nl_init() is equivalent to

setlocale(LC_ALL, "");

and is supplied for compatibility with X/Open XPG2.

RETURN VALUES

If a valid string is given for the *locale* parameter, and the selection can be honored, **setlocale**() returns the string associated with the specified *category* for the new locale. If the selection cannot be honored, **setlocale**() returns a null pointer and the program's locale is not changed.

A NULL pointer for *locale* causes **setlocale**() to return the string associated with the *category* for the program's current locale; the program's locale is not changed. The string contains information relating to each piece part of the whole international environment. This inquiry can fail by returning a null pointer if any *category* is invalid.

The string returned by such a **setlocale**() call is such that a subsequent call with the string and its associated category will restore that part of the program's locale. The string returned by:

ptr = setlocale(LC_ALL, (char *) 0);

is such that in a subsequent call:

setlocale(LC_ALL, ptr);

will reset each and every category to the state when the string was first returned. The string returned must not be modified by the program, but will be overwritten by a subsequent call to **setlocale**().

FILES

/etc/locale/*locale***/***category*
: *locale* is the directory that contains numerous files (*categories*), each relating to a single category of a valid *locale* as selected by category argument to **setlocale**(). Generally this is classed as a private directory. This directory is searched by **setlocale**(), prior to searching:

/usr/share/lib/locale/*locale***/***category*
: *locale* is the directory that contains numerous files (*categories*), each relating to a single category of a valid *locale* as selected by category argument to **setlocale**(). Generally this data is classed as global and sharable.

DIAGNOSTICS

setlocale( ) returns a null pointer if a relevant environment variable has an invalid setting. **setlocale( )** also returns a null pointer if *category* is invalid.

NAME

setuid, seteuid, setruid, setgid, setegid, setrgid – set user and group ID

SYNOPSIS

#include <sys/types.h>

int setuid(uid)
uid_t uid;

int seteuid(euid)
uid_t euid;

int setruid(ruid)
uid_t ruid;

int setgid(gid)
gid_t gid;

int setegid(egid)
gid_t egid;

int setrgid(rgid)
gid_t rgid;

DESCRIPTION

setuid( ) (**setgid( )**) sets both the real and effective user ID (group ID) of the current process as specified by *uid* (*gid*) (see NOTES).

seteuid( ) (**setegid( )**) sets the effective user ID (group ID) of the current process.

setruid( ) (**setrgid( )**) sets the real user ID (group ID) of the current process.

These calls are only permitted to the super-user or if the argument is the real or effective user (group) ID of the calling process.

SYSTEM V DESCRIPTION

If the effective user ID of the calling process is not super-user, but if its real user (group) ID is equal to *uid* (*gid*), or if the saved set-user (group) ID from **execve**(2V) is equal to *uid* (*gid*), then the effective user (group) ID is set to *uid* (*gid*).

RETURN VALUES

These functions return:

0 on success.

–1 on failure and set **errno** to indicate the error as for **setreuid**(2) (**setregid**(2)).

ERRORS

EINVAL The value of *uid* (*gid*) is invalid (less than 0 or greater than 65535).

EPERM The process does not have super-user privileges and *uid* (*gid*) does not matches neither the real user (group) ID of the process nor the saved set-user-ID (set-group-ID) of the process.

SEE ALSO

execve(2V), **getgid**(2V), **getuid**(2V), **setregid**(2), **setreuid**(2)

NOTES

For **setuid( )** to behave as described above, {_POSIX_SAVED_IDS} must be in effect (see **sysconf**(2V)). {_POSIX_SAVED_IDS} is always in effect on SunOS systems, but for portability, applications should call **sysconf( )** to determine whether {_POSIX_SAVED_IDS} is in effect for the current system.

NAME

sigaction – examine and change signal action

SYNOPSIS

```
#include <signal.h>

int sigaction(sig, act, oact)
int sig;
struct sigaction *act, *oact;
```

DESCRIPTION

sigaction( ) allows the calling process to examine and specify (or both) the action to be associated with a specific signal. *sig* specifies the signal. Acceptable values are defined in **<signal.h>**.

The structure **sigaction**( ), used to describe an action to be taken, is defined in the header **<signal.h>** as follows:

```
struct sigaction {
        void (*sa_handler)();      /* SIG_DFL, SIG_IGN, or pointer to a function */
        sigset_t sa_mask;          /* Additional signals to be blocked during
                                      execution of signal-catching function */
        int sa_flags;              /* Special flags to affect behavior of signal */
};
```

If **act** is not NULL, it points to a structure specifying the action to be associated with the specified signal. If **oact** is not NULL, the action previously associated with the signal is stored in the location pointed to by the **oact.** If **act** is NULL, signal handling is unchanged by this function. Thus, the call can be used to enquire about the current handling of a given signal. The **sa_handler** field of the **sigaction** structure identifies the action to be associated with the specified signal. If the **sa_handler** field specifies a signal-catching function, the **sa_mask** field identifies a set of signals that shall be added to the process's signal mask before the signal-catching function mask is invoked. The SIGKILL and SIGSTOP signals shall not be added to the signal mask using this mechanism; this restriction shall be enforced by the system without causing an error to be indicated.

The **sa_flags** field can be used to modify the behavior of the specified signal. The following flag bit, defined in the header **<signal.h>**, can be set in **sa_flags:**

```
#define SA_ONSTACK        0x0001 /* take signal on signal stack */
#define SA_INTERRUPT      0x0002 /* do not restart system on signal return */
#define SA_RESETHAND      0x0004 /* reset handler to SIG_DFL when signal taken */
#define SA_NOCLDSTOP      0x0008 /* don't send a SIGCHLD on child stop */
```

If *sig* is **SIGCHILD** and the **SA_NOCLDSTOP** flag is not set in **sa_flags**, and the implementation supports the **SIGCHILD** signal, a **SIGCHILD** signal shall be generated for the calling process whenever any of its child processes stop. If *sig* is **SIGCHILD** and the **SA_NOCLDSTOP** flag is set in **sa_flags**, the implementation shall not generate a **SIGCHILD** signal in this way.

If the **SA_ONSTACK** bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with **sigstack**(2), rather than delivering the signal on the current stack.

If a caught signal occurs during certain system calls, the call is restarted by default. The call can be forced to terminate prematurely with an EINTR error return by setting the **SA_INTERRUPT** bit in the flags for that signal. **SA_INTERRUPT** is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected system calls are **read**(2V) or **write**(2V) on a slow device (such as a terminal or pipe or other socket, but not a file) and during a **wait**(2V).

Once a signal handler is installed, it remains installed until another **sigvec**( ) call is made, or an **execve**(2V) is performed, unless the **SA_RESETHAND** bit is set in the flags for that signal. In that case, the value of the handler for the caught signal is set to **SIG_DFL** before entering the signal-catching function, unless the signal is **SIGILL** or **SIGTRAP**. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The **SA_RESETHAND** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

When a signal is caught by a signal-catching function installed by **sigaction()** a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either **sigprocmask()** or **sigsuspend()**). This mask is formed by taking the union of the current signal mask and the value of the **sa_mask** for the signal being delivered, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to **sigaction()**), or until one of the **exec** functions is called.

If the previous action for *sig* had been established by **signal()** defined in the C standard, the values of the fields returned in the structure pointed to by the **oact** are unspecified, and in particular **oact->sv_handler** is not necessarily the same value passed to **signal()**. However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to **sigaction()** using **act**, handling of the signal shall be as if the original call to **signal()** were repeated.

If **sigaction()** fails, no new signal handler is installed.

RETURN VALUES

sigaction() returns:

0 on success.

−1 on failure and sets **errno** to indicate the error.

ERRORS

EINVAL *sig* is an invalid or unsupported signal number.

An attempt was made to catch a signal that cannot be ignored. See **<signal.h>**.

SEE ALSO

kill(2V), **sigpause**(2V), **sigprocmask**(2V), **signal**(3V), **sigsetops**(3V)

NAME

sigfpe – signal handling for specific SIGFPE codes

SYNOPSIS

#include <signal.h>

#include <floatingpoint.h>

sigfpe_handler_type sigfpe(code, hdl)
sigfpe_code_type code;
sigfpe_handler_type hdl;

DESCRIPTION

This function allows signal handling to be specified for particular **SIGFPE** codes. A call to **sigfpe()** defines a new handler *hdl* for a particular **SIGFPE** *code* and returns the old handler as the value of the function **sigfpe()** . Normally handlers are specified as pointers to functions; the special cases **SIGFPE_IGNORE**, **SIGFPE_ABORT**, and **SIGFPE_DEFAULT** allow ignoring, specifying core dump using **abort**(3), or default handling respectively.

For these IEEE-related codes:

FPE_FLTINEX_TRAP	fp_inexact - floating inexact result
FPE_FLTDIV_TRAP	fp_division - floating division by zero
FPE_FLTUND_TRAP	fp_underflow - floating underflow
FPE_FLTOVF_TRAP	fp_overflow - floating overflow
FPE_FLTBSUN_TRAP	fp_invalid - branch or set on unordered
FPE_FLTOPERR_TRAP	fp_invalid - floating operand error
FPE_FLTNAN_TRAP	fp_invalid - floating Not-A-Number

default handling is defined to be to call the handler specified to **ieee_handler**(3M).

For all other **SIGFPE** codes, default handling is to core dump using **abort**(3).

The compilation option **–ffpa** causes fpa recomputation to replace the default abort action for code **FPE_FPA_ERROR**. Note: **SIGFPE_DEFAULT** will restore abort rather than FPA recomputation for this code.

Three steps are required to intercept an IEEE-related **SIGFPE** code with **sigfpe()**:

1) Set up a handler with **sigfpe()**.

2) Enable the relevant IEEE trapping capability in the hardware, perhaps by using assembly-language instructions.

3) Perform a floating-point operation that generates the intended IEEE exception.

Unlike **ieee_handler**(3M), **sigfpe()** never changes floating-point hardware mode bits affecting IEEE trapping. No IEEE-related **SIGFPE** signals will be generated unless those hardware mode bits are enabled.

SIGFPE signals can be handled using **sigvec**(2), **signal**(3V), **sigfpe**(3), or **ieee_handler**(3M). In a particular program, to avoid confusion, use only one of these interfaces to handle **SIGFPE** signals.

EXAMPLE

A user-specified signal handler might look like this:

```
void sample_handler( sig, code, scp, addr )
        int sig ;            /* sig == SIGFPE always */
        int code ;
        struct sigcontext *scp ;
        char *addr ;
        {
                /*
                   Sample user-written sigfpe code handler.
                   Prints a message and continues.
                   struct sigcontext is defined in <signal.h>.
                 */
                printf(" ieee exception code %x occurred at pc %X \n",code,scp->sc_pc);
        }
```

and it might be set up like this:

```
        extern void sample_handler();
        main()
        {
                sigfpe_handler_type hdl, old_handler1, old_handler2;
        /*
         * save current overflow and invalid handlers; set the new
         * overflow handler to sample_handler() and set the new
         * invalid handler to SIGFPE_ABORT (abort on invalid)
         */
                hdl = (sigfpe_handler_type) sample_handler;
                old_handler1 = sigfpe(FPE_FLTOVF_TRAP, hdl);
                old_handler2 = sigfpe(FPE_FLTOPERR_TRAP, SIGFPE_ABORT);
                ...
        /*
         * restore old overflow and invalid handlers
         */
                sigfpe(FPE_FLTOVF_TRAP,    old_handler1);
                sigfpe(FPE_FLTOPERR_TRAP, old_handler2);
        }
```

SEE ALSO

sigvec(2), **abort**(3), **floatingpoint**(3), **ieee_handler**(3M), **signal**(3V)

DIAGNOSTICS

sigfpe( ) returns BADSIG if *code* is not zero or a defined SIGFPE code.

NAME

siginterrupt – allow signals to interrupt system calls

SYNOPSIS

int siginterrupt(sig, flag)
int sig, flag;

DESCRIPTION

siginterrupt( ) is used to change the system call restart behavior when a system call is interrupted by the specified signal. If the flag is false (0), then system calls will be restarted if they are interrupted by the specified signal and no data has been transferred yet. System call restart is the default behavior on 4.2BSD, and on SunOS in the 4.2 environment, when the **signal (3V)** routine is used.

If the flag is true (1), then restarting of system calls is disabled. If a system call is interrupted by the specified signal and no data has been transferred, the system call will return −1 with **errno** set to EINTR. Interrupted system calls that have started transferring data will return the amount of data actually transferred. System call interrupt is the signal behavior found on older version of the UNIX operating systems, such as 4.1BSD and System V UNIX. It is the default behavior on SunOS in the System V environment when the **signal()** routine is used; therefore, this routine is useful in that environment only if a signal that a **sigvec**(2) specified should restart system calls is to be changed not to restart them.

Note: the new 4.2BSD signal handling semantics are not altered in any other way. Most notably, signal handlers always remain installed until explicitly changed by a subsequent **sigvec**( ) call, and the signal mask operates as documented in **sigvec**( ), unless the **SV_RESETHAND** bit has been used to specify that the pre-4.2BSD signal behavior is to be used. Programs may switch between restartable and interruptible system call operation as often as desired in the execution of a program.

Issuing a **siginterrupt**( ) call during the execution of a signal handler will cause the new action to take place on the next signal to be caught.

NOTES

This library routine uses an extension of the **sigvec**(2) system call that is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUES

siginterrupt( ) returns:

0 on success.

−1 if an invalid signal number was supplied.

SEE ALSO

sigblock(2), **sigpause**(2V), **sigsetmask**(2), **sigvec**(2), **signal**(3V)

NAME

signal – simplified software signal facilities

SYNOPSIS

```
#include <signal.h>

void (*signal(sig, func))()
void (*func)();
```

DESCRIPTION

signal() is a simplified interface to the more general **sigvec**(2) facility. Programs that use **signal()** in preference to **sigvec()** are more likely to be portable to all systems.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see **termio**(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the **SIGKILL** and **SIGSTOP** signals, the **signal()** call allows signals either to be ignored or to interrupt to a specified location. The following is a list of all signals with names as in the include file **<signal.h>**:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGABRT	6*	abort (generated by **abort**(3) routine)
SIGEMT	7*	emulator trap
SIGFPE	8*	arithmetic exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe or other socket with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16•	urgent condition present on socket
SIGSTOP	17†	stop (cannot be caught, blocked, or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19•	continue after stop
SIGCHLD	20•	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGIO	23•	I/O is possible on a descriptor (see **fcntl**(2V))
SIGXCPU	24	cpu time limit exceeded (see **getrlimit**(2))
SIGXFSZ	25	file size limit exceeded (see **getrlimit**(2))
SIGVTALRM	26	virtual time alarm (see **getitimer**(2))
SIGPROF	27	profiling timer alarm (see **getitimer**(2))
SIGWINCH	28•	window changed (see **termio**(4) and **win**(4S))
SIGLOST	29*	resource lost (see **lockd**(8C))
SIGUSR1	30	user-defined signal 1
SIGUSR2	31	user-defined signal 2

The starred signals in the list above cause a core image if not caught or ignored.

If *func* is **SIG_DFL**, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If *func* is **SIG_IGN** the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and *func* is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. **Unlike previous signal facilities, the handler *func* remains installed after a signal has been delivered.**

If a caught signal occurs during certain system calls, terminating the call prematurely, the call is automatically restarted. In particular this can occur during a **read**(2V) or **write**(2V) on a slow device (such as a terminal; but not a file) and during a **wait**(2V).

The value of **signal**( ) is the previous (or initial) value of *func* for the particular signal.

After a **fork**(2V) or **vfork**(2) the child inherits all signals. An **execve**(2V) resets all caught signals to the default action; ignored signals remain ignored.

SYSTEM V DESCRIPTION

If *func* is **SIG_IGN** the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs, *func* is called. Further occurrences of the signal are not automatically blocked. The value of *func* for the caught signal is reset to **SIG_DFL** before *func* is called, unless the signal is **SIGILL** or **SIGTRAP**.

A return from the function continues the process at the point at which it was interrupted. The handler *func* does not remain installed after a signal has been delivered.

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is interrupted. In particular this can occur during a **read**(2V) or **write**(2V) on a slow device (such as a terminal; but not a file) and during a **wait**(2V). After the signal catching function returns, the interrupted system call may return a −1 to the calling process with **errno** set to EINTR.

RETURN VALUES

signal( ) returns the previous action on success. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

signal( ) will fail and no action will take place if one of the following occurs:

EINVAL	*sig* was not a valid signal number.
	An attempt was made to ignore or supply a handler for **SIGKILL** or **SIGSTOP**.

SEE ALSO

kill(1), **execve**(2V), **fork**(2V), **getitimer**(2), **getrlimit**(2), **kill**(2V), **ptrace**(2), **read**(2V), **sigblock**(2), **sigpause**(2V), **sigsetmask**(2), **sigstack**(2), **sigvec**(2), **vfork**(2), **wait**(2V), **write**(2V), **setjmp**(3V), **termio**(4)

NOTES

The handler routine can be declared:

```
void handler(sig, code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **<signal.h>**), used to restore the context from before the signal; and *addr* is additional address information. See **sigvec**(2) for more details.

NAME

sigsetops, sigaddset, sigdelset, sigfillset, sigemptyset, sigismember – manipulate signal sets

SYNOPSIS

```
#include <signal.h>

int sigaddset(set, signo)
sigset_t *set;
int signo;

int sigdelset(set, signo)
sigset_t *set;
int signo;

int sigfillset(set)
sigset_t *set;

int sigemptyset(set)
sigset_t *set;

int sigismember(set, signo)
sigset_t *set
int signo;
```

DESCRIPTION

The **sigsetops** primitives manipulate sets of signals. They operate on data objects addressable by the application. They do not operate on any set of signals known to the system, such as the set blocked from delivery to a process or the set pending for a process.

sigaddset() and **sigdelset()** respectively add and delete the individual signal specified by the value of **signo** from the signal set pointed to by *set*.

sigemptyset() initializes the signal set pointed to by *set* such that all signals defined in this standard are excluded.

sigfillset() initializes the signal set pointed to by *set* such that all signals defined in this standard are included.

Applications shall call either **sigemptyset()** or **sigfillset()** at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of **sigaddset()**, **sigdelset()**, **sigismember()**, **sigaction()**, **sigprocmask()**, **sigpending()**, or **sigsuspend()** the results are undefined.

sigismember() tests whether the signal specified by the value of **signo** is a member of the set pointed to by *set*.

RETURN VALUES

sigismember() returns:

1 if the specified signal is a member of *set*.

0 if the specified signal is not a member of *set*.

−1 if an error is detected, and sets **errno** to indicate the error.

The other functions return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

For each of the following conditions, if the condition is detected, **sigaddset()**, **sigdelset()**, and **sigismember()** set **errno** to:

EINVAL **signo** is an invalid or unsupported signal number.

SEE ALSO

sigaction(3V), **sigpending**(2V), **sigprocmask**(2V)

NAME

sleep – suspend execution for interval

SYNOPSIS

int sleep(seconds)
unsigned seconds;

SYSTEM V SYNOPSIS

unsigned sleep(seconds)
unsigned seconds;

DESCRIPTION

sleep() suspends the current process from execution for the number of seconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system.

sleep() is implemented by setting an interval timer and pausing until it expires. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous value of the timer, the process sleeps only until the timer would have expired, and the signal which occurs with the expiration of the timer is sent one second later.

SYSTEM V DESCRIPTION

sleep() suspends the current process from execution until either the number of real time seconds specified by *seconds* have elapsed or a signal is delivered to the calling process and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be an arbitrary amount longer than requested because of other activity in the system. The value returned by **sleep()** will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested **sleep()** time, or premature arousal due to another caught signal.

RETURN VALUES

sleep() returns no useful value.

SYSTEM V RETURN VALUES

If **sleep()** returns because the requested time has elapsed, it returns 0. If **sleep()** returns due to the delivery of a signal, it returns the "unslept" amount in seconds.

SEE ALSO

getitimer(2), **sigpause**(2V), **usleep**(3)

NOTES

SIGALRM should *not* be blocked or ignored during a call to **sleep()**. Only a prior call to **alarm**(3V) should generate SIGALRM for the calling process during a call to **sleep()**. A signal-catching function should *not* interrupt a call to **sleep()** to call **siglongjmp()** or **longjmp()** to restore an environment saved prior to the **sleep()** call.

WARNINGS

sleep() is slightly incompatible with **alarm**(3V). Programs that do not execute for at least one second of clock time between successive calls to **sleep()** indefinitely delay the alarm signal. Use System V **sleep()**. Each **sleep(3V)** call postpones the alarm signal that would have been sent during the requested sleep period to occur one second later.

NAME

sputl, sgetl – access long integer data in a machine-independent fashion

SYNOPSIS

void sputl(value, buffer)
long value;
char *buffer;

long sgetl(buffer)
char *buffer;

DESCRIPTION

sputl() takes the four bytes of the long integer **values** and places them in memory starting at the address pointed to by *buffer*. The ordering of the bytes is the same across all machines.

sgetl() retrieves the four bytes in memory starting at the address pointed to by *buffer* and returns the long integer value in the byte ordering of the host machine.

The combination of **sputl()** and **sgetl()** provides a machine-independent way of storing long numeric data in a file in binary form without conversion to characters.

NAME

ssignal, gsignal – software signals

SYNOPSIS

#include <signal.h>

int (*ssignal (sig, action))( )
int sig, (*action)( );

int gsignal (sig)
int sig;

DESCRIPTION

ssignal( ) and **ssignal**( ) implement a software facility similar to **signal**(3V).

Software signals made available to users are associated with integers in the inclusive range 1 through 15. A call to **ssignal**( ) associates a procedure, *action*, with the software signal *sig*; the software signal, *sig*, is raised by a call to **ssignal**( ). Raising a software signal causes the action established for that signal to be *taken*.

The first argument to **ssignal**( ) is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) *action function* or one of the manifest constants **SIG_DFL (default)**or **SIG_IGN (ignore). ssignal**( ) returns the action previously established for that signal type; if no action has been established or the signal number is illegal, **ssignal**( ) returns **SIG_DFL.**

ssignal( ) raises the signal identified by its argument, *sig*:

> If an action function has been established for *sig*, then that action is reset to **SIG_DFL** and the action function is entered with argument *sig*. **ssignal**( ) returns the value returned to it by the action function.
>
> If the action for *sig* is **SIG_IGN, ssignal**( ) returns the value 1 and takes no other action.
>
> If the action for *sig* is **SIG_DFL, ssignal**( ) returns the value 0 and takes no other action.
>
> If *sig* has an illegal value or no action was ever specified for *sig*, **ssignal**( ) returns the value 0 and takes no other action.

SEE ALSO

signal(3V)

NAME

stdio – standard buffered input/output package

SYNOPSIS

#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;

DESCRIPTION

The functions described in section 3S constitute a user-level I/O buffering scheme. The in-line macros **getc**(3V) and **putc**(3S) handle characters quickly. The macros **getchar**( ) (see **getc**(3V)) and **putchar**( ) (see **putc**(3S)), and the higher level routines **fgetc**( ), **getw**( ) (see **getc**(3V)), **gets**(3S), **fgets**( ) (see **gets**(3S)), **scanf**(3V), **fscanf**( ) (see **scanf**(3V)), **fread**(3S), **fputc**( ), **putw**( ) (see **putc**(3S)), **puts**(3S), **fputs**( ) (see **puts**(3S)), **printf**(3V), **fprintf**( ) (see **printf**(3V)), **fwrite**( ) (see **fread**(3S)) all use or act as if they use **getc**( ) and **putc**( ). They can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type FILE. **fopen**(3V) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant NULL (0) designates a nonexistent pointer.

An integer constant EOF (–1) is returned upon EOF or error by most integer functions that deal with streams (see the individual descriptions for details).

Any module that uses this package must include the header file of pertinent macro definitions, as follows:

#include <stdio.h>

The functions and constants mentioned in sections labeled 3S of this manual are declared in that header file and need no further declaration. The constants and the following 'functions' are implemented as macros; redeclaration of these names is perilous: **getc**( ), **getchar**( ), **putc**( ), **putchar**( ), **feof**( ), **ferror**( ), **fileno**( ), and **clearerr**( ).

Output streams, with the exception of the standard error stream **stderr**, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream **stderr** is by default unbuffered, but use of **fopen**( ) will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is written to the destination file or terminal as soon as it is output to the stream; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is written to the destination file or terminal as soon as the line is completed (that is, as soon as a NEWLINE character is output or, if the output stream is **stdout** or **stderr**, as soon as input is read from **stdin**). **setbuf**(3V), **setbuffer**( ), **setlinebuf**( ), or **setvbuf**( ) (see **setbuf**(3V)) can be used to change the stream's buffering strategy.

SYSTEM V DESCRIPTION

When an output stream is line-buffered, each line of output is written to the destination file or terminal as soon as the line is completed (that is, as soon as a NEWLINE character is output or as soon as input is read from a line-buffered stream).

Output saved up on *all* line-buffered streams is written when input is read from *any* line-buffered stream. Input read from a stream that is not line-buffered does not flush output on line-buffered streams.

RETURN VALUES

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with **fopen**(), input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

SEE ALSO

open(2V), **close**(2V), **lseek**(2V), **pipe**(2V), **read**(2V), **write**(2V), **ctermid**(3V), **cuserid**(3V), **fclose**(3V), **ferror**(3V), **fopen**(3V), **fread**(3S), **fseek**(3S), **getc**(3V), **gets**(3S), **popen**(3S), **printf**(3V), **putc**(3S), **puts**(3S), **scanf**(3V), **setbuf**(3V), **system**(3), **tmpfile**(3S), **tmpnam**(3S), **ungetc**(3S)

NOTES

The line buffering of output to terminals is almost always transparent, but may cause confusion or malfunctioning of programs which use standard I/O routines but use **read**(2V) to read from the standard input, as calls to **read**() do not cause output to line-buffered streams to be flushed.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to call **fflush**() (see **fclose**(3V)) on the standard output before performing the computation so that the output will appear.

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially **vfork**(2).

NAME

strcoll, strxfrm – compare or transform strings using collating information

SYNOPSIS

```
#include <string.h>

int strcoll(s1, s2)
char *s1;
char *s2;

size_t strxfrm(s1, s2, n)
char *s1;
char *s2;
size_t n;
```

DESCRIPTION

strcoll() compares the string pointed to by *s1* to the string pointed to by *s2*. These strings are interpreted as appropriate to the **LC_COLLATE** category of the current locale.

strxfrm() transforms the string pointed to by *s2* and places the resulting string into the array pointed to by *s1*. The transformation is such that if **string()** is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the **strcoll()** function applied to the same two original strings. No more than *n* characters are placed into the resulting array pointed to by *s1*, including the terminating null character. If *n* is zero, *s1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUES

On success, **strcoll()** returns an integer greater than, equal to or less than zero, respectively, if the string pointed to by *s1* is greater than, equal to or less than the string pointed to by *s2* when both are interpreted as appropriate to the current locale. On failure, **strcoll()** sets **errno** to indicate the error, but returns no special value.

strxfrm() returns the length of the transformed string, not including the terminating null character. If the value returned is *n* or more, the contents of the array pointed to by *s1* are indeterminate. On failure, **strxfrm()** returns **(size_t)**–1, and sets **errno** to indicate the error.

ERRORS

EINVAL *s1* or *s2* contain characters outside the domain of the collating sequence.

SEE ALSO

string(3)

NAME

strcat, strncat, strdup, strcmp, strncmp, strcasecmp, strncasecmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, strtok, index, rindex – string operations

SYNOPSIS

#include <string.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;
int n;

char *strdup(s1)
char *s1;

int strcmp(s1, s2)
char *s1, *s2;

int strncmp(s1, s2, n)
char *s1, *s2;
int n;

int strcasecmp(s1, s2) char *s1, *s2;

int strncasecmp(s1, s2, n)
char *s1, *s2;
int n;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;
int n;

int strlen(s)
char *s;

char *strchr(s, c)
char *s;
int c;

char *strrchr(s, c)
char *s;
int c;

char *strpbrk(s1, s2)
char *s1, *s2;

int strspn(s1, s2)
char *s1, *s2;

int strcspn(s1, s2)
char *s1, *s2;

char *strstr(s1, s2)
char *s1, *s2;

char *strtok(s1, s2)
char *s1, *s2;

#include <strings.h>

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

strcat() appends a copy of string *s2* to the end of string *s1*. **strncat()** appends at most *n* characters. Each returns a pointer to the null-terminated result.

strcmp() compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*. **strncmp()** makes the same comparison but compares at most *n* characters. Two additional routines **strcasecmp()** and **strncasecmp()** compare the strings and ignore differences in case. These routines assume the ASCII character set when equating lower and upper case characters.

strdup() returns a pointer to a new string which is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using **malloc**(3V). If the new string cannot be created, a NULL pointer is returned.

strcpy() copies string *s2* to *s1* until the null character has been copied. **strncpy()** copies string *s2* to *s1* until either the null character has been copied or *n* characters have been copied. If the length of *s2* is less than *n*, **strncpy()** pads *s1* with null characters. If the length of *s2* is *n* or greater, *s1* will not be null-terminated. Both functions return *s1*.

strlen() returns the number of characters in *s*, not including the null-terminating character.

strchr() (**strrchar()**) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

index() (**rindex()**) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. These functions are identical to **strchr()** (**strchr()**) and merely have different names.

strpbrk() returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

strspn() (**strcspn()**) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

strstr() returns a pointer to the first occurrence of the pattern string *s2* in *s1*. For example, if *s1* is **"string thing"** and *s2* is **"ing"**, **strstr()** returns **"ing thing"**. If *s2* does not occur in *s1*, **strstr()** returns NULL.

strtok() considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

NOTES

For user convenience, all these functions, except for **index()** and **rindex()**, are declared in the optional **<string.h>** header file. All these functions, including **index()** and **rindex()** but excluding **strchr()**, **strrchr()**, **strpbrk()**, **strspn()**, **strcspn()**, and **strtok()** are declared in the optional **<strings.h>** include file; these headers are set this way for backward compatibility.

SEE ALSO

malloc(3V), **bstring**(3)

WARNINGS

strcmp() and **strncmp()** use native character comparison, which is signed on the Sun, but may be unsigned on other machines. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

strcasecmp() and **strncasecmp()** use native character comparison as above and assume the *ASCII* character set.

On the Sun processor, as well as on many other machines, you can *not* use a NULL pointer to indicate a null string. A NULL pointer is an error and results in an abort of the program. If you wish to indicate a null string, you must have a pointer that points to an explicit null string. On some implementations of the C language on some machines, a NULL pointer, if dereferenced, would yield a null string; this highly non-portable trick was used in some programs. Programmers using a NULL pointer to represent an empty string should be aware of this portability issue; even on machines where dereferencing a NULL pointer does not cause an abort of the program, it does not necessarily yield a null string.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

string_to_decimal, file_to_decimal, func_to_decimal – parse characters into decimal record

SYNOPSIS

```
#include <floatingpoint.h>
#include <stdio.h>

void string_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;

void file_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar,pf,pnread)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;
FILE *pf;
int *pnread;

void func_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar,pget,pnread,punget)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;
int (*pget)( );
int *pnread;
int (*punget)( );
```

DESCRIPTION

The **char_to_decimal( )** functions parse a numeric token from at most *nmax* characters in a string ***pc* or file **pf* or function *(*pget)( )* into a decimal record **pd*, classifying the form of the string in **pform* and **pechar*. The accepted syntax is intended to be sufficiently flexible to accomodate many languages:

whitespace value

or

whitespace sign value

where *whitespace* is any number of characters defined by *isspace* in **<ctype.h>**, *sign* is either of [+–], and *value* can be *number*, *nan*, or *inf*. *inf* can be INF (*inf_form*) or INFINITY (*infinity_form*) without regard to case. *nan* can be NAN (*nan_form*) or NAN(*nstring*) (*nanstring_form*) without regard to case; *nstring* is any string of characters not containing ')' or the null character; *nstring* is copied to *pd*–>**ds** and, currently, not used subsequently. *number* consists of

significant

or

significant efield

where *significant* must contain one or more digits and may contain one point; possible forms are

digits	*(int_form)*
digits.	*(intdot_form)*
.digits	*(dotfrac_form)*
digits.digits	*(intdotfrac_form)*

efield consists of

echar digits

or

echar sign digits

where *echar* is one of [Ee], and *digits* contains one or more digits.

When *fortran_conventions* is nonzero, additional input forms are accepted according to various Fortran conventions:

0 no Fortran conventions
1 Fortran list-directed input conventions
2 Fortran formatted input conventions, ignore blanks **(BN)**
3 Fortran formatted input conventions, blanks are zeros **(BZ)**

When *fortran_conventions* is nonzero, *echar* may also be one of [Dd], and *efield* may also have the form

sign digits.

When *fortran_conventions*>= 2, blanks may appear in the *digits* strings for the integer, fraction, and exponent fields and may appear between *echar* and the exponent sign and after the infinity and NaN forms. If *fortran_conventions*== 2, the blanks are ignored. When *fortran_conventions*== 3, the blanks that appear in *digits* strings are interpreted as zeros, and other blanks are ignored.

When *fortran_conventions* is zero, the current locale's decimal point character is used as the decimal point; when *fortran_conventions* is nonzero, the period is used as the decimal point.

The form of the accepted decimal string is placed in **peform*. If an *efield* is recognized, **pechar* is set to point to the *echar*.

On input, **pc* points to the beginning of a character string buffer of length >= *nmax*. On output, **pc* points to a character in that buffer, one past the last accepted character. **string_to_decimal( )** gets its characters from the buffer; **file_to_decimal( )** gets its characters from **pf* and records them in the buffer, and places a null after the last character read. **func_to_decimal( )** gets its characters from an int function *(*pget)*( ).

The scan continues until no more characters could possibly fit the acceptable syntax or until *nmax* characters have been scanned. If the *nmax* limit is not reached then at least one extra character will usually be scanned that is not part of the accepted syntax. **file_to_decimal( )** and **func_to_decimal( )** set **pnread* to the number of characters read from the file; if greater than *nmax*, some characters were lost. If no characters were lost, **file_to_decimal( )** and **func_to_decimal( )** attempt to push back, with **ungetc**(3S) or *(*punget)*( ), as many as possible of the excess characters read, adjusting **pnread* accordingly. If all unget calls are successful, then ***pc* will be a null character. No push back will be attempted if *(*punget)*( ) is NULL.

Typical declarations for **pget*( ) and **punget*( ) are:

```
int xget( )
{ ... }
int (*pget)( ) = xget;
int xunget(c)
char c ;
{ ... }
int (*punget)( ) = xunget;
```

If no valid number was detected, *pd*->**fpclass** is set to **fp_signaling**, **pc* is unchanged, and **pform* is set to **invalid_form.**

atof( ) and **strtod**(3) use **string_to_decimal**( ). **scanf**(3V) uses **file_to_decimal**( ).

SEE ALSO

ctype(3V), **localeconv**(3), **scanf**(3V), **setlocale**(3V), **strtod**(3), **ungetc**(3S)

NAME

strtod, atof – convert string to double-precision number

SYNOPSIS

double strtod(str, ptr)
char *str, **ptr;

double atof(str)
char *str;

DESCRIPTION

strtod( ) returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character, using **string_to_decimal**(3), with *fortran_conventions* set to 0.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, **ptr* is set to *str*, and for historical compatibility, 0.0 is returned, although a NaN would better match the IEEE Floating-Point Standard's intent.

The radix character is defined by the program's locale (category LC_NUMERIC). In the "C" locale, or in a locale where the radix character is not defined. the radix character defaults to a period '.'.

atof(str) is equivalent to **strtod(str, (char **)NULL)**. Thus, when **atof(str)** returns 0.0 there is no way to determine whether *str* contained a valid numerical string representing 0.0 or an invalid numerical string.

SEE ALSO

scanf(3V), **string_to_decimal**(3)

DIAGNOSTICS

Exponent overflow and underflow produce the results specified by the IEEE Standard. In addition, **errno** is set to ERANGE.

NAME

strtol, atol, atoi – convert string to integer

SYNOPSIS

long strtol(str, ptr, base)
char *str, **ptr;
int base;

long atol(str)
char *str;

int atoi(str)
char *str;

DESCRIPTION

strtol( ) returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading ''white-space'' characters (as defined by **isspace( )** in **ctype**(3V)) are ignored.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, that location is set to *str*, and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and ''0x'' or ''0X'' is ignored if *base* is 16.

If *base* is zero, the string itself determines the base thusly: after an optional leading sign a leading zero indicates octal conversion, and a leading ''0x'' or ''0X'' hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

atol(*str*) is equivalent to **strtol**(*str*, (*char* **)NULL, **10**).

atoi(*str*) is equivalent to (int) **strtol**(str, (char **)NULL, **10**).

SEE ALSO

ctype(3V), **scanf**(3V), **strtod**(3)

BUGS

Overflow conditions are ignored.

NAME

stty, gtty – set and get terminal state

SYNOPSIS

```
#include <sgtty.h>

stty(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;
```

DESCRIPTION

Note: this interface is obsoleted by **ioctl**(2).

stty() sets the state of the terminal associated with *fd*. **stty()** retrieves the state of the terminal associated with *fd*. To set the state of a terminal the call must have write permission.

The **stty()** call is actually

ioctl(fd, TIOCSETP, buf)

while the **gtty()** call is

ioctl(fd, TIOCGETP, buf)

See **ioctl**(2) and **ttcompat**(4M) for an explanation.

DIAGNOSTICS

If the call is successful 0 is returned, otherwise –1 is returned and the global variable **errno** contains the reason for the failure.

SEE ALSO

ioctl(2), **ttcompat**(4M)

NAME

swab – swap bytes

SYNOPSIS

void
swab(from, to, nbytes)
char *from, *to;

DESCRIPTION

swab() copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between high-ender machines (IBM 360's, MC68000's, etc) and low-end machines (such as Sun386i systems).

nbytes should be even and positive. If *nbytes* is odd and positive, **swab()** uses *nbytes* − 1 instead. If *nbytes* is negative, **swab()** does nothing.

The *from* and *to* addresses should not overlap in portable programs.

NAME

syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logopt, facility)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

setlogmask(maskpri)

DESCRIPTION

syslog() passes *message* to **syslogd**(8), which logs it in an appropriate system log, writes it to the system console, forwards it to a list of users, or forwards it to the **syslogd** on another host over the network. The message is tagged with a priority of *priority*. The message looks like a **printf**(3V) string except that **%m** is replaced by the current error message (collected from **errno**). A trailing NEWLINE is added if needed.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from an ordered list:

LOG_EMERG	A panic condition. This is normally broadcast to all users.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions, such as hard device errors.
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

If special processing is needed, **openlog()** can be called to initialize the log file. The parameter *ident* is a string that is prepended to every message. *logopt* is a bit field indicating logging options. Current values for *logopt* are:

LOG_PID	Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork).
LOG_CONS	Write messages to the system console if they cannot be sent to **syslogd**. This option is safe to use in daemon processes that have no controlling terminal, since **syslog()** forks before opening the console.
LOG_NDELAY	Open the connection to **syslogd** immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.
LOG_NOWAIT	Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using **SIGCHLD**, since **syslog()** may otherwise block waiting for a child whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded:

LOG_KERN	Messages generated by the kernel. These cannot be generated by any user processes.
LOG_USER	Messages generated by random user processes. This is the default facility identifier if none is specified.
LOG_MAIL	The mail system.
LOG_DAEMON	System daemons, such as **ftpd**(8C), **routed**(8C), etc.
LOG_AUTH	The authorization system: **login**(1), **su**(1V), **getty**(8), etc.
LOG_LPR	The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc.
LOG_NEWS	Reserved for the USENET network news system.
LOG_UUCP	Reserved for the UUCP system; it does not currently use **syslog**.
LOG_CRON	The **cron/at** facility; **crontab**(1), **at**(1), **cron**(8), etc.
LOG_LOCAL0–7	Reserved for local use.

closelog( ) can be used to close the log file.

setlogmask( ) sets the log priority mask to *maskpri* and returns the previous mask. Calls to **syslog**( ) with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro **LOG_MASK**(*pri*); the mask for all priorities up to and including *toppri* is given by the macro **LOG_UPTO**(*toppri*). The default allows all priorities to be logged.

EXAMPLES

This call logs a message at priority **LOG_ALERT**:

syslog(LOG_ALERT, "who: internal error 23");

The FTP daemon **ftpd** would make this call to **openlog**( ) to indicate that all messages it logs should have an identifying string of **ftpd**, should be treated by **syslogd** as other messages from system daemons are, should include the process ID of the process logging the message:

openlog("ftpd", LOG_PID, LOG_DAEMON);

Then it would make the following call to **setlogmask**( ) to indicate that messages at priorities from **LOG_EMERG** through **LOG_ERR** should be logged, but that no messages at any other priority should be logged:

setlogmask(LOG_UPTO(LOG_ERR));

Then, to log a message at priority **LOG_INFO**, it would make the following call to **syslog**:

syslog(LOG_INFO, "Connection from host %d", CallingHost);

A locally-written utility could use the following call to **syslog**( ) to log a message at priority **LOG_INFO** to be treated by **syslogd** as other messages to the facility **LOG_LOCAL2** are:

syslog(LOG_INFO|LOG_LOCAL2, "error: %m");

SEE ALSO

at(1), **crontab**(1), **logger**(1), **login**(1), **lpr**(1), **su**(1V), **printf**(3V), **syslog.conf**(5), **cron**(8), **ftpd**(8C), **getty**(8), **lpc**(8), **lpd**(8), **routed**(8C), **syslogd**(8)

NAME

system – issue a shell command

SYNOPSIS

system(string)
char *string;

DESCRIPTION

system( ) gives the *string* to **sh**(1) as input, just as if the string had been typed as a command from a terminal. The current process performs a **wait**(2V) system call, and waits until the shell terminates. **system( )** then returns the exit status returned by **wait**(2V). Unless the shell was interrupted by a signal, its termination status is contained in the 8 bits higher up from the low-order 8 bits of the value returned by **wait( )**.

SEE ALSO

sh(1), **execve**(2V), **wait**(2V), **popen**(3S)

DIAGNOSTICS

Exit status 127 (may be displayed as "32512") indicates the shell could not be executed.

NAME

t_accept – accept a connect request

SYNOPSIS

```
#include <tiuser.h>

int t_accept(fd, resfd, call)
int fd;
int resfd;
struct t_call *call;
```

DESCRIPTION

t_accept() is issued by a transport user to accept a connect request. *fd* identifies the local transport endpoint where the connect indication arrived, *resfd* specifies the local transport endpoint where the connection is to be established, and *call* contains information required by the transport provider to complete the connection. *call* points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

The *netbuf* structure contains the following members:

```
unsigned int maxlen;
unsigned int len;
char *buf;
```

buf points to a user input and/or output buffer. *len* generally specifies the number of bytes contained in the buffer. If the structure is used for both input and output, the transport function will replace the user value of *len* on return. *maxlen* generally has significance only when *buf* is used to receive output from the transport function. In this case, it specifies the physical size of the buffer, and the maximum value of *len* that can be set by the function. If *maxlen* is not large enough to hold the returned information, a TBUFOVFLW error will generally result. However, certain functions may return part of the data and not generate an error. In *call*, *addr* is the address of the caller, *opt* indicates any protocol-specific parameters associated with the connection, *udata* points to any user data to be returned to the caller, and *sequence* is the value returned by **t_listen**(3N) that uniquely associates the response with a previously received connect indication.

A transport user may accept a connection on either the same, or on a different, local transport endpoint than the one on which the connect indication arrived. If the same endpoint is specified (*resfd* = *fd*), the connection can be accepted unless the following condition is true: The user has received other indications on that endpoint but has not responded to them (with **t_accept()** or **t_snddis**(3N)). For this condition, **t_accept()** will fail and set **t_errno** to TBADF.

If a different transport endpoint is specified (*resfd* != *fd*), the endpoint must be bound to a protocol address and must be in the **T_IDLE** state (see **t_getstate**(3N)) before the **t_accept()** is issued.

For both types of endpoints, **t_accept()** will fail and set **t_errno** to TLOOK if there are indications (such as a connect or disconnect) waiting to be received on that endpoint.

The values of parameters specified by *opt* and the syntax of those values are protocol specific. The *udata* field enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned by **t_open**(3N) or **t_getinfo**(3N). If the *len* field of *udata* is zero, no data will be sent to the caller.

RETURN VALUES

t_accept() returns:

0 on success.

–1 on failure and sets **t_errno** to indicate the error.

ERRORS

TACCES	The user does not have permission to accept a connection on the responding transport endpoint.
	The user does not have permission to use the specified options.
TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.
TBADF	The specified file descriptor does not refer to a transport endpoint.
	The user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived.
TBADOPT	The specified options were in an incorrect format or contained illegal information.
TBADSEQ	An invalid sequence number was specified.
TLOOK	An asynchronous event has occurred on the transport endpoint referenced by *fd* and requires immediate attention.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
	The transport endpoint referred to by *resfd* is not in the **T_IDLE** state.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_connect**(3N), **t_getstate**(3N), **t_listen**(3N), **t_open**(3N), **t_rcvconnect**(3N)

Network Programming

NAME

t_alloc – allocate a library structure

SYNOPSIS

```
#include <tiuser.h>

char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;
```

DESCRIPTION

t_alloc() dynamically allocates memory for the various transport function argument structures as specified below. **t_alloc()** allocates memory for the specified structure and for buffers referenced by the structure.

The structure to allocate is specified by *struct_type*, and can be one of the following (each of of these structures may be used as an argument to one or more transport functions):

T_BIND	**struct t_bind**
T_CALL	**struct t_call**
T_OPTMGMT	**struct t_optmgmt**
T_DIS	**struct t_discon**
T_UNITDATA	**struct t_unitdata**
T_UDERROR	**struct t_uderr**
T_INFO	**struct t_info**

Each of the above structures, except **T_INFO**, contains at least one field of type '**struct netbuf**'. The *maxlen*, *len*, and *buf* members of the **netbuf** structure are described in **t_accept**(3N). For each field of this type, the user may specify that the buffer for that field should be allocated as well. The *fields* argument specifies this option, where the argument is the bitwise-OR of any of the following:

T_ADDR	The *addr* field of the **t_bind**, **t_call**, **t_unitdata**, or **t_uderr** structures.
T_OPT	The *opt* field of the **t_optmgmt**, **t_call**, **t_unitdata**, or **t_uderr** structures.
T_UDATA	The *udata* field of the **t_call**, **t_discon**, or **t_unitdata** structures.
T_ALL	All relevant fields of the given structure.

For each field specified in *fields*, **t_alloc()** allocates memory for the buffer associated with the field, and initializes the *buf* pointer and *maxlen* field accordingly. The length of the buffer allocated is based on the same size information returned to the user on **t_open**(3N) and **t_getinfo**(3N). Thus, *fd* must refer to the transport endpoint through which the newly allocated structure is passed, so that the appropriate size information can be accessed. If the size value associated with any specified field is –1 or –2 (see **t_open**(3N) or **t_getinfo**(3N)), **t_alloc()** is unable to determine the size of the buffer to allocate and fails, setting **t_errno** to TSYSERR and **errno** to EINVAL . For any field not specified in *fields*, *buf* is set to NULL and *maxlen* is set to zero.

Use of **t_alloc()** to allocate structures helps ensure the compatibility of user programs with future releases of the transport interface.

RETURN VALUES

On success, **t_alloc()** returns a pointer to the type of structure specified by **struct_type**. On failure, it returns NULL and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_free**(3N), **t_getinfo**(3N), **t_open**(3N)

Network Programming

NAME

t_bind – bind an address to a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;

DESCRIPTION

t_bind() associates a protocol address with the transport endpoint specified by *fd* and activates that transport endpoint. In connection mode, the transport provider may begin accepting or requesting connections on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint.

The *req* and *ret* arguments point to a **t_bind**() structure containing the following members:

struct netbuf addr;
unsigned qlen;

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). The *addr* field of the **t_bind**() structure specifies a protocol address and the *qlen* field is used to indicate the maximum number of outstanding connect indications.

req is used to request that an address, represented by the *netbuf* structure, be bound to the given transport endpoint. *len* specifies the number of bytes in the address and *buf* points to the address buffer. *maxlen* has no meaning for the *req* argument. On return, *ret* contains the address that the transport provider actually bound to the transport endpoint; this may be different from the address specified by the user in *req*. In *ret*, the user specifies *maxlen* which is the maximum size of the address buffer and *buf* which points to the buffer where the address is to be placed. On return, *len* specifies the number of bytes in the bound address and *buf* points to the bound address. If *maxlen* is not large enough to hold the returned address, an error will result.

If the requested address is not available, or if no address is specified in *req* (the *len* field of *addr* in *req* is 0) the transport provider will assign an appropriate address to be bound, and will return that address in the *addr* field of *ret*. The user can compare the addresses in *req* and *ret* to determine whether the transport provider bound the transport endpoint to a different address than that requested.

req may be NULL if the user does not wish to specify an address to be bound. Here, the value of *qlen* is assumed to be 0, and the transport provider must assign an address to the transport endpoint. Similarly, *ret* may be NULL if the user does not care what address was bound by the transport provider and is not interested in the negotiated value of *qlen*. It is valid to set *req* and *ret* to NULL for the same call, in which case the transport provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The *qlen* field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connect indications the transport provider should support for the given transport endpoint. An outstanding connect indication is one that has been passed to the transport user by the transport provider. A value of *qlen* greater than 0 is only meaningful when issued by a passive transport user that expects other users to call it. The value of *qlen* will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connect indications. On return, the *qlen* field in *ret* will contain the negotiated value.

t_bind() allows more than one transport endpoint to be bound to the same protocol address (however, the transport provider must support this capability also), but binding more than one protocol address to the same transport endpoint is not allowed. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect indications associated with that protocol address. In other words, only one **t_bind**() for a given protocol address may specify a value of *qlen* greater than 0. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connect indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of *qlen* greater than 0, the transport provider will assign another address to be bound to that endpoint. If a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of that connection. No other transport endpoints may be bound for listening while that initial listening endpoint is in the data transfer phase. This will prevent more than one transport endpoint bound to the same protocol address from accepting connect indications.

RETURN VALUES

t_bind() returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TACCES	The user does not have permission to use the specified address.
TBADADDR	The specified protocol address was in an incorrect format or contained illegal information.
TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The transport provider's state will change to **T_IDLE** and the information to be returned in *ret* will be discarded.
TNOADDR	The transport provider could not allocate an address.
TOUTSTATE	The function was issued in the wrong sequence.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_open**(3N), **t_optmgmt**(3N), **t_unbind**(3N)

Network Programming

NAME

t_close – close a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_close(fd)
int fd;

DESCRIPTION

t_close() informs the transport provider that the user is finished with the transport endpoint specified by *fd*, and frees any local library resources associated with the endpoint. In addition, **t_close()** closes the file associated with the transport endpoint.

t_close() should be called from the **T_UNBND** state (see **t_getstate**(3N)). However, **t_close()** does not check state information, so it may be called from any state to close a transport endpoint. If this occurs, the local library resources associated with the endpoint will be freed automatically. In addition, **close**(2V) will be issued for that file descriptor; the close will be abortive if no other process has that file open, and will break any transport connection that may be associated with that endpoint.

RETURN VALUES

t_close() returns:

0	on success.
−1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.

SEE ALSO

close(2V), **t_getstate**(3N), **t_open**(3N), **t_unbind**(3N)

Network Programming

NAME

t_connect – establish a connection with another transport user

SYNOPSIS

```
#include <tiuser.h>

int t_connect(fd, sndcall, rcvcall)
int fd;
struct t_call *sndcall;
struct t_call *rcvcall;
```

DESCRIPTION

t_connect() enables a transport user to request a connection to the specified destination transport user. *fd* identifies the local transport endpoint where communication will be established, while *sndcall* and *rcvcall* point to a **t_call()** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

sndcall specifies information needed by the transport provider to establish a connection and *rcvcall* specifies information that is associated with the newly established connection.

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). In *sndcall*, *addr* specifies the protocol address of the destination transport user, *opt* presents any protocol-specific information that might be needed by the transport provider, *udata* points to optional user data that may be passed to the destination transport user during connection establishment, and *sequence* has no meaning for this function.

On return in *rcvcall*, *addr* returns the protocol address associated with the responding transport endpoint, *opt* presents any protocol-specific information associated with the connection, *udata* points to optional user data that may be returned by the destination transport user during connection establishment, and *sequence* has no meaning for this function.

opt implies no structure on the options that may be passed to the transport provider. The transport provider is free to specify the structure of any options passed to it. These options are specific to the underlying protocol of the transport provider. The user may choose not to negotiate protocol options by setting the *len* field of *opt* to 0. In this case, the transport provider may use default options.

udata enables the caller to pass user data to the destination transport user and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned by **t_open**(3N) or **t_getinfo**(3N). If the *len* field of *udata* is 0 in *sndcall*, no data will be sent to the destination transport user.

On return, the *addr*, *opt*, and *udata* fields of *rcvcall* will be updated to reflect values associated with the connection. Thus, the *maxlen* field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, *rcvcall* may be NULL in which case no information is given to the user on return from **t_connect()**.

By default, **t_connect()** executes in synchronous mode, and will wait for the destination user's response before returning control to the local user. A successful return (a return value of 0) indicates that the requested connection has been established. However, if **T_NDELAY** is set (using **t_open()** or **fcntl**), **t_connect()** executes in asynchronous mode. In this case, the call will not wait for the remote user's response, but will return control immediately to the local user and return −1 with **t_errno** set to TNODATA to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connect request to the destination transport user.

RETURN VALUES

t_connect() returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TACCES	The user does not have permission to use the specified address or options.
TBADADDR	The specified protocol address was in an incorrect format or contained illegal information.
TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.
TBADF	The specified file descriptor does not refer to a transport endpoint.
TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.
TBUFOVFLW	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. If executed in synchronous mode, the transport provider's state, as seen by the user, changes to **T_DATAXFER** and the connect indication information to be returned in *rcvcall* is discarded.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA	**T_NDELAY** was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The function was issued in the wrong sequence.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_accept**(3N), **t_getinfo**(3N), **t_listen**(3N), **t_open**(3N), **t_optmgmt**(3N), **t_rcvconnect**(3N)

Network Programming

NAME

t_error – produce error message

SYNOPSIS

```
#include <tiuser.h>

void t_error(errmsg)
char *errmsg;

extern int t_errno;
extern char *t_errlist[ ];
extern int t_nerr;
```

DESCRIPTION

t_error( ) produces a message on the standard error output which describes the last error received during a call to a transport function. The argument string *errmsg* is a user-supplied error message that gives context to the error. **t_error( )** prints the user-supplied error message followed by a colon and a standard error message for the current error defined in **t_errno**. To simplify variant formatting of messages, the array of message strings **t_errlist** is provided; **t_errno** can be used as an index in this table to get the message string without the NEWLINE. **t_nerr** is the largest message number provided for in the **t_errlist** table.

t_errno is only set when an error occurs and is not cleared on successful calls.

EXAMPLE

If a **t_connect**(3N) function fails on transport endpoint *fd*2 because a bad address was given, the following call might follow the failure:

```
t_error ("t_connect failed on fd2");
```

The diagnostic message to be printed would look like:

```
t_connect failed on fd2:  Incorrect transport address format
```

where '**Incorrect transport address format**' identifies the specific error that occurred, and '**t_connect failed on fd2**' tells the user which function failed on which transport endpoint.

SEE ALSO

Network Programming

NAME

t_free – free a library structure

SYNOPSIS

```
#include <tiuser.h>

int t_free(ptr, struct_type)
char *ptr;
int struct_type;
```

DESCRIPTION

t_free( ) frees memory previously allocated by **t_alloc**(3N). This function will free memory for the specified structure, and will also free memory for buffers referenced by the structure.

ptr points to one of the six structure types described for **t_alloc**(3N), and *struct_type* identifies the type of that structure which can be one of the following:

T_BIND	**struct t_bind**
T_CALL	**struct t_call**
T_OPTMGMT	**struct t_optmgmt**
T_DIS	**struct t_discon**
T_UNITDATA	**struct t_unitdata**
T_UDERROR	**struct t_uderr**
T_INFO	**struct t_info**

where each of these structures is used as an argument to one or more transport functions.

t_free( ) checks the *addr*, *opt*, and *udata* fields of the given structure (as appropriate), and frees the buffers pointed to by the *buf* field of the *netbuf* (see **intro**(3)) structure. The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). If *buf* is NULL, **t_free**( ) will not attempt to free memory. After all buffers are freed, **t_free**( ) will free the memory associated with the structure pointed to by *ptr*.

Undefined results will occur if *ptr* or any of the *buf* pointers points to a block of memory that was not previously allocated by **t_alloc**(3N).

RETURN VALUES

t_free( ) returns:

0 on success.

–1 on failure and sets **t_errno** to indicate the error.

ERRORS

TSYSERR The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_alloc**(3N)

Network Programming

NAME

t_getinfo – get protocol-specific service information

SYNOPSIS

```
#include <tiuser.h>

int t_getinfo(fd, info)
int fd;
struct t_info *info;
```

DESCRIPTION

t_getinfo( ) returns the current characteristics of the underlying transport protocol associated with file descriptor *fd*. The *info* structure is used to return the same information returned by **t_open**(3N). **t_getinfo( )** enables a transport user to access this information during any phase of communication.

This argument points to a **t_info** structure which contains the following members:

```
long addr;       /* max size of the transport protocol address */
long options;    /* max number of bytes of protocol-specific options */
long tsdu;       /* max size of a transport service data unit (TSDU) */
long etsdu;      /* max size of an expedited transport service data unit (ETSDU) */
long connect;    /* max amount of data allowed on connection establishment
                        functions */
long discon;     /* max amount of data allowed on t_snddis and t_rcvdis functions */
long servtype;   /* service type supported by the transport provider */
```

FIELDS

The values of the fields have the following meanings:

addr　A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of –1 specifies that there is no limit on the address size; and a value of –2 specifies that the transport provider does not provide user access to transport protocol addresses.

options　A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of –1 specifies that there is no limit on the option size; and a value of –2 specifies that the transport provider does not support user-settable options.

tsdu　A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of a TSDU; and a value of –2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu　A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of an ETSDU; and a value of –2 specifies that the transfer of expedited data is not supported by the transport provider.

connect　A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of –1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of –2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the **t_snddis**(3N) and **t_rcvdis**(3N) functions; a value of −1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of −2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the **t_alloc**(3N) function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of option negotiation, and **t_getinfo**( ) enables a user to retrieve the current characteristics.

RETURN VALUES

The *servtype* field of *info* may specify one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports a connectionless-mode service. For this service type, **t_open**(3N) will return −2 for the **etsdu**, **connect**, and **discon** fields.

RETURN VALUES

t_getinfo( ) returns 0 on success and −1 on failure.

ERRORS

TBADF The specified file descriptor does not refer to a transport endpoint.

TSYSERR The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N)

Network Programming

NAME

t_getstate – get the current state

SYNOPSIS

#include <tiuser.h>

int t_getstate(fd)
int fd;

DESCRIPTION

t_getstate() returns the current state of the provider associated with the transport endpoint specified by *fd*.

If the provider is undergoing a state transition when **t_getstate()** is called, the function will fail. **t_getstate()** returns the current state on successful completion and −1 on failure and **t_errno** is set to indicate the error. The current state may be one of the following:

T_UNBND	unbound
T_IDLE	idle
T_OUTCON	outgoing connection pending
T_INCON	incoming connection pending
T_DATAXFER	data transfer
T_OUTREL	outgoing orderly release (waiting for an orderly release indication)
T_INREL	incoming orderly release (waiting for an orderly release request)

RETURN VALUES

t_getstate() returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TSTATECHNG	The transport provider is undergoing a state change.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N)

Network Programming

NAME

t_listen – listen for a connect request

SYNOPSIS

```
#include <tiuser.h>

int t_listen(fd, call)
int fd;
struct t_call *call;
```

DESCRIPTION

t_listen( ) listens for a connect request from a calling transport user. *fd* identifies the local transport endpoint where connect indications arrive, and on return, *call* contains information describing the connect indication. *call* points to a **t_call**( ) structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). In *call*, *addr* returns the protocol address of the calling transport user, *opt* returns protocol-specific parameters associated with the connect request, *udata* returns any user data sent by the caller on the connect request, and *sequence* is a number that uniquely identifies the returned connect indication. The value of *sequence* enables the user to listen for multiple connect indications before responding to any of them.

Since this function returns values for the *addr*, *opt*, and *udata* fields of *call*, the *maxlen* field of each must be set before issuing the **t_listen**( ) to indicate the maximum size of the buffer for each.

By default, **t_listen**( ) executes in synchronous mode and waits for a connect indication to arrive before returning to the user. However, if **T_NDELAY** is set (using **t_open**(3N) or **fcntl**( )), **t_listen**( ) executes asynchronously, reducing to a **poll**(2) for existing connect indications. If none are available, it returns −1 and sets **t_errno** to TNODATA.

RETURN VALUES

t_listen( ) returns:

0	on success.
−1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to **T_INCON** and the connect indication information to be returned in *call* is discarded.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA	**T_NDELAY** was set, but no connect indications had been queued.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_accept**(3N), **t_bind**(3N), **t_connect**(3N), **t_open**(3N), **t_rcvconnect**(3N)

Network Programming

NAME

t_look – look at the current event on a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_look(fd)
int fd;

DESCRIPTION

t_look() returns the current event on the transport endpoint specified by *fd*. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, TLOOK, on the current or next function to be executed.

This function also enables a transport user to **poll**(2) a transport endpoint periodically for asynchronous events.

RETURN VALUES

Upon success, **t_look**() returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists. One of the following events is returned:

T_LISTEN	Connection indication received
T_CONNECT	Connect confirmation received
T_DATA	Normal data received
T_EXDATA	Expedited data received
T_DISCONNECT	Disconnect received
T_ERROR	Fatal error indication
T_UDERR	Datagram error indication
T_ORDREL	Orderly release indication

On failure, −1 is returned and **t_errno** is set to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N)

Network Programming

NAME

t_open – establish a transport endpoint

SYNOPSIS

```
#include <tiuser.h>

int t_open(path, oflag, info)
char *path;
int oflag;
struct t_info *info;
```

DESCRIPTION

t_open() must be called as the first step in the initialization of a transport endpoint. It establishes a transport endpoint by opening a file that identifies a particular transport provider (such as a transport protocol) and returning a file descriptor that identifies that endpoint. For example, opening the file **/dev/tcp** identifies an OSI connection-oriented transport layer protocol as the transport provider. Currently, **/dev/tcp** is the only transport protocol available to **t_open()**.

path points to the pathname of the file to open, and *oflag* identifies any open flags (as in **open**(2V)). **t_open()** returns a file descriptor that will be used by all subsequent functions to identify the particular local transport endpoint.

This function also returns various default characteristics of the underlying transport protocol by setting fields in the **t_info** structure pointed to by *info*. **t_info** is defined in **<nettli/tiuser.h>** as:

```
struct t_info {
        long addr;      /* size of protocol address */
        long options;   /* size of protocol options */
        long tsdu;      /* size of max transport service data unit */
        long etsdu;     /* size of max expedited tsdu */
        long connect;   /* max data for connection primitives */
        long discon;    /* max data for disconnect primitives */
        long servtype;  /* provider service type */
};
```

The fields of this structure have the following values:

addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of –1 specifies that there is no limit on the address size; and a value of –2 specifies that the transport provider does not provide user access to transport protocol addresses.

options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of –1 specifies that there is no limit on the option size; and a value of –2 specifies that the transport provider does not support user-settable options.

tsdu A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of a TSDU; and a value of –2 specifies that the transfer of normal data is not supported by the transport provider.

etdsu A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of an ETSDU; and a value of –2 specifies that the transfer of expedited data is not supported by the transport provider.

connect A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of −1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of −2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the **t_snddis**(3N) and **t_rcvdis**(3N) functions; a value of −1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of −2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype This field specifies the service type supported by the transport provider.

The *servtype* field of *info* may specify one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports a connectionless-mode service. For this service type, **t_open()** will return −2 for *etsdu*, *connect*, and *discon*.

A single transport endpoint may support only one of the above services at one time.

If *info* is set to NULL by the transport user, no protocol information is returned by **t_open()**.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the **t_alloc**(3N) function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.

RETURN VALUES

t_open() returns a non-negative file descriptor on success. On failure, it returns −1 and sets **t_errno** to indicate the error.

ERRORS

TSYSERR The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

open(2V), **tcp**(4P)

Network Programming

NAME

t_optmgmt – manage options for a transport endpoint

SYNOPSIS

#include <tiuser.h>

```
int t_optmgmt(fd, req, ret)
int fd;
struct t_optmgmt *req;
struct t_optmgmt *ret;
```

DESCRIPTION

t_optmgmt( ) enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. *fd* identifies a bound transport endpoint.

The *req* and *ret* arguments point to a **t_optmgmt**( ) structure containing the following members:

```
struct netbuf opt;
long    flags;
```

The *opt* field identifies protocol options and the *flags* field is used to specify the action to take with those options.

The options are represented by a *netbuff* structure in a manner similar to the address in **t_bind**(3N). The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). *req* is used to request a specific action of the provider and to send options to the provider. *len* specifies the number of bytes in the options, *buf* points to the options buffer, and *maxlen* has no meaning for the *req* argument. The transport provider may return options and flag values to the user through *ret*. For *ret*, *maxlen* specifies the maximum size of the options buffer and *buf* points to the buffer where the options are to be placed. On return, *len* specifies the number of bytes of options returned. *maxlen* has no meaning for the *req* argument, but must be set in the *ret* argument to specify the maximum number of bytes the options buffer can hold. The actual structure and content of the options is imposed by the transport provider.

The **flags** field of *req* can specify one of the following actions:

T_NEGOTIATE Enables the user to negotiate the values of the options specified in *req* with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through *ret*.

T_CHECK Enables the user to verify whether the options specified in *req* are supported by the transport provider. On return, the **flags** field of *ret* will have either **T_SUCCESS** or **T_FAILURE** set to indicate to the user whether the options are supported. These flags are only meaningful for the **T_CHECK** request.

T_DEFAULT Enables a user to retrieve the default options supported by the transport provider into the *opt* field of *ret*. In *req*, the *len* field of *opt* must be zero and the *buf* field may be NULL.

If issued as part of the connectionless-mode service, **t_optmgmt**( ) may block due to flow control constraints. **t_optmgmt**( ) will not complete until the transport provider has processed all previously sent data units.

RETURN VALUES

t_optmgmt( ) returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TACCES	The user does not have permission to negotiate the specified options.
TBADF	The specified file descriptor does not refer to a transport endpoint.
TBADFLAG	An invalid flag was specified.
TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.
TBUFOVFLW	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in *ret* will be discarded.
TOUTSTATE	The function was issued in the wrong sequence.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_getinfo**(3N), **t_open**(3N)

Network Programming

NAME

t_rcv – receive normal or expedited data sent over a connection

SYNOPSIS

int t_rcv(fd, buf, nbytes, flags)

int fd;
char *buf;
unsigned nbytes;
int *flags;

DESCRIPTION

t_rcv() receives either normal or expedited data. *fd* identifies the local transport endpoint through which data will arrive, *buf* points to a receive buffer where user data will be placed, and *nbytes* specifies the size of the receive buffer. *flags* may be set on return from **t_rcv()** and specifies optional flags as described below.

By default, **t_rcv()** operates in synchronous mode and will wait for data to arrive if none is currently available. However, if **T_NDELAY** is set (using **t_open**(3N) or **fcntl**()), **t_rcv()** will execute in asynchronous mode and will fail if no data is available. See TNODATA below.

On return from the call, if **T_MORE** is set in *flags* this indicates that there is more data and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple **t_rcv()** calls. Each **t_rcv()** with the **T_MORE** flag set indicates that another **t_rcv()** must follow immediately to get more data for the current TSDU. The end of the TSDU is identified by the return of a **t_rcv()** call with the **T_MORE** flag not set. If the transport provider does not support the concept of a TSDU as indicated in the *info* argument on return from **t_open**(3N) or **t_getinfo**(3N), the **T_MORE** flag is not meaningful and should be ignored.

On return, the data returned is expedited data if **T_EXPEDITED** is set in *flags*. If the number of bytes of expedited data exceeds *nbytes*, **t_rcv()** will set **T_EXPEDITED** and **T_MORE** on return from the initial call. Subsequent calls to retrieve the remaining ETSDU will not have **T_EXPEDITED** set on return. The end of the ETSDU is identified by the return of a **t_rcv()** call with the **T_MORE** flag not set.

If expedited data arrives after part of a TSDU has been retrieved, receipt of the remainder of the TSDU will be suspended until the ETSDU has been processed. Only after the full ETSDU has been retrieved (T_MORE not set) will the remainder of the TSDU be available to the user.

RETURN VALUES

On success, **t_rcv()** returns the number of bytes received. On failure, it returns –1.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA	**T_NDELAY** was set, but no data is currently available from the transport provider.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N), **t_snd**(3N)

Network Programming

NAME

t_rcvconnect – receive the confirmation from a connect request

SYNOPSIS

```
#include <tiuser.h>

int t_rcvconnect(fd, call)
int fd;
struct t_call *call;
```

DESCRIPTION

t_rcvconnect allows a calling transport user to get the status of a previous connect request. It can be used in conjunction with **t_connect**(3N) to establish a connection in asynchronous mode.

fd identifies the local transport endpoint where communication is established. *call* contains information associated with the newly established connection *call* points to a *t_call* structure that contains information associated with the new connection, and is defined in **<nettli/tiuser.h>** as:

```
struct t_call {
        struct netbuf addr;
        struct netbuf opt;
        struct netbuf udata;
        int sequence;
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). In the *t_call* structure, *addr* returns the protocol address associated with the responding transport endpoint, *opt* presents protocol-specific information associated with the connection, *udata* points to optional user data that may be returned by the destination transport user during connection establishment, and *sequence* has no meaning for this function.

The *maxlen* field of each argument must be set before issuing this function to indicate the maximum buffer size. However, *call* may be NULL, in which case no information is given to the user on return from **t_rcvconnect**( ). By default, **t_rcvconnect**( ) executes synchronously and waits for the connection before returning. On return, the *addr*, *opt*, and *udata* fields reflect values associated with the connection.

If **O_NDELAY** is set (using **t_open**(3N) or **fcntl**( )), **t_rcvconnect**( ) executes asynchronously, reducing to a **poll**(2) request for existing connect confirmations. If none are available, **t_rcvconnect**( ) fails and returns immediately without waiting for the connection to be established. See **TNODATA** below. **t_rcvconnect**( ) must be re-issued at a later time to complete the connection establishment phase and retrieve the information returned in *call*.

RETURN VALUES

t_rcvconnect( ) returns:

0　　on success.

−1　　on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The bytes allocated for an incoming argument is sufficient to store the value of that argument and the connect information to be returned in *call* is discarded. The transport provider's state, as seen by the user, will be changed to DATAXFER.
TNODATA	**O_NDELAY** was set, but a connect confirmation has not yet arrived.
TLOOK	An asynchronous event has occurred on this transport connection and requires immediate attention.

TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

poll(2), **intro**(3), **t_accept**(3N), **t_bind**(3N), **t_connect**(3N), **t_listen**(3N), **t_open**(3N)

Network Programming

NAME

t_rcvdis – retrieve information from disconnect

SYNOPSIS

```
#include <tiuser.h>

t_rcvdis(fd, discon)
int fd;
struct t_discon *discon;
```

DESCRIPTION

t_rcvdis( ) is used to identify the cause of a disconnect, and to retrieve any user data sent with the disconnect. *fd* identifies the local transport endpoint where the connection existed, and *discon* points to a **t_discon** structure defined in **<nettli/tiuser.>** as:

```
struct t_discon {
        struct netbuf udata;          /* user data */
        int reason;                   /* reason code */
        int sequence;                 /* sequence number */
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). *reason* specifies the reason for the disconnect through a protocol-dependent reason code, *udata* identifies any user data that was sent with the disconnect, and *sequence* may identify an outstanding connect indication with which the disconnect is associated. *sequence* is only meaningful when **t_rcvdis( )** is issued by a passive transport user who has executed one or more **t_listen(3N)** functions and is processing the resulting connect indications. If a disconnect indication occurs, *sequence* can be used to identify which of the outstanding connect indications is associated with the disconnect.

If a user does not care if there is incoming data and does not need to know the value of *reason* or *sequence*, *discon* may be NULL and any user data associated with the disconnect will be discarded. However, if a user has retrieved more than one outstanding connect indication (using **t_listen**(3N)) and *discon* is NULL, the user will be unable to identify with which connect indication the disconnect is associated.

RETURN VALUES

t_rcvdis( ) returns:

0	on success.
–1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The number of bytes allocated for incoming data is not sufficient to store the data. The provider's state, as seen by the user, will change to **T_IDLE** and the disconnect indication information to be returned in *discon* will be discarded.
TNODIS	No disconnect indication currently exists on the specified transport endpoint.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_connect**(3N), **t_listen**(3N), **t_open**(3N), **t_snddis**(3N)

Network Programming

NAME

t_rcvrel – acknowledge receipt of an orderly release indication

SYNOPSIS

#include <tiuser.h>

int t_rcvrel(fd)
int fd;

DESCRIPTION

t_rcvel( ) acknowledges receipt of an orderly release indication. *fd* identifies the local transport endpoint where the connection exists. After receipt of this indication, the user may not attempt to receive more data because such an attempt will block forever. However, the user may continue to send data over the connection if **t_sndrel**(3N) has not been issued by the user.

t_rcvrel( ) is an optional service of the transport provider, and is only supported if the transport provider returned service type **T_COTS_ORD** on **t_open**(3N) or **t_getinfo**(3N).

RETURN VALUES

t_rcvrel( ) returns:

0	on success.
−1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNOREL	No orderly release indication currently exists on the specified transport endpoint.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N), **t_sndrel**(3N)

Network Programming

NAME

t_rcvudata – receive a data unit

SYNOPSIS

```
#include <tiuser.h>

int t_rcvudata(fd, unitdata, flags)
int fd;
struct t_unitdata *unitdata;
int *flags;
```

DESCRIPTION

t_rcvudata( ) is used in connectionless mode to receive a data unit from another transport user. *fd* identifies the local transport endpoint through which data will be received, *unitdata* holds information associated with the received data unit, and *flags* is set on return to indicate that the complete data unit was not received. *unitdata* points to a *t_unitdata* structure defined in **<nettli/tiuser.h>** as:

```
struct t_unitdata {
        struct netbuf addr;        /* address      */
        struct netbuf opt;         /* options      */
        struct netbuf udata;       /* user data    */
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). The *maxlen* field of *addr*, *opt*, and *udata* must be set before issuing **t_rcvudata**( ) to indicate the maximum size of the buffer for each.

On return from this call, *addr* specifies the protocol address of the sending user, *opt* identifies protocol-specific options that were associated with this data unit, and *udata* specifies the user data that was received.

By default, **t_rcvudata**( ) operates in synchronous mode and will wait for a data unit to arrive if none is currently available. However, if **O_NDELAY** is set (using **t_open**(3N) or **fcntl**( )), **t_rcvudata**( ) will execute in asynchronous mode and will fail if no data units are available.

If the buffer defined in the *udata* field of *unitdata* is not large enough to hold the current data unit, the buffer will be filled and **T_MORE** will be set in *flags* on return to indicate that another **t_rcvudata**( ) should be issued to retrieve the rest of the data unit. Subsequent **t_rcvudata**( ) call(s) will return zero for the length of the address and options until the full data unit has been received.

RETURN VALUES

t_rcvudata( ) returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data information to be returned in *unitdata* will be discarded.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA	**T_NDELAY** was set, but no data units are currently available from the transport provider.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_rcvuderr**(3N), **t_sndudata**(3N)

NAME

t_rcvuderr – receive a unit data error indication

SYNOPSIS

#include <tiuser.h>

```
int t_rcvuderr(fd, uderr)
int fd;
struct t_uderr *uderr;
```

DESCRIPTION

t_rcvuderr( ) is used in connectionless mode to receive information concerning an error on a previously sent data unit, and should only be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. *fd* identifies the local transport endpoint through which the error report will be received, and *uderr* points to a **t_uderr**( ) structure defined in **<nettli/tiuser.h>** as:

```
struct t_uderr {
        struct netbuf addr;          /* address    */
        struct netbuf opt;           /* options    */
        long  error;                 /* error code */
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). The *maxlen* field of *addr* and *opt* must be set before issuing this function to indicate the maximum size of the buffer for each.

On return from this call, the *addr* structure specifies the destination protocol address of the erroneous data unit, the *opt* structure identifies protocol-specific options that were associated with the data unit, and **error** specifies a protocol-dependent error code.

If the user does not care to identify the data unit that produced an error, *uderr* may be set to NULL and **t_rcvuderr**( ) will simply clear the error indication without reporting any information to the user.

RETURN VALUES

t_rcvuderr( ) returns:

0 on success.

–1 on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data error information to be returned in *uderr* will be discarded.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TNOUDERR	No unit data error indication currently exists on the specified transport endpoint.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_rcvudata**(3N), **t_sndudata**(3N)

Network Programming

NAME

t_snd – send normal or expedited data over a connection

SYNOPSIS

```
#include <tiuser.h>

int t_snd(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int flags;
```

DESCRIPTION

t_snd() sends either normal or expedited data. *fd* identifies the local transport endpoint over which data should be sent, *buf* points to the user data, *nbytes* specifies the number of user data bytes to be sent, and *flags* specifies any optional flags described below.

By default, **t_snd()** operates synchronously and may wait if flow control restrictions prevents data acceptance by the local transport provider when the call is made. However, if **O_NDELAY** is set (using **t_open**(3N) or **fcntl()**), **t_snd()** executes asynchronously, and fails immediately if there are flow control restrictions.

On success, **t_snd()** returns the byte total accepted by the transport provider. This normally equals the bytes total specified in *nbytes*. If **O_NDELAY** is set, it is possible that the transport provider will accept only part of the data. In this case, **t_snd()** will set **T_MORE** for the data that was sent (see below) and returns a value less than *nbytes*. If *nbytes* is zero, no data is passed to the provider; **t_snd()** returns zero.

If **T_EXPEDITED** is set in *flags*, the data is sent as expedited data, subject to the interpretations of the transport provider.

T_MORE indicates to the transport provider that the transport service data unit (TSDU), or expedited transport service data unit (ETSDU), is being sent through multiple **t_snd()** calls. In these calls, the **T_MORE** flag indicates another **t_snd()** is to follow; the end of TSDU (or ETSDU) is identified by a **t_snd()** call without the **T_MORE** flag. **T_MORE** allows the sender to break up large logical data units, while preserving their boundaries at the other end. The flag does not imply how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the *info* argument on return from **t_open**(3N) or **t_getinfo**(3N), the **T_MORE** flag is meaningless.

The size of each TSDU or ETSDU must not exceed the transport provider limits as returned by **t_open**(3N) or **t_getinfo**(3N). Failure to comply results in protocol error EPROTO. See TSYSERR below.

If **t_snd()** is issued from the **T_IDLE** state, the provider may silently discard the data. If **t_snd()** is issued from any state other than **T_DATAXFER** or **T_IDLE** the provider generates a EPROTO error.

RETURN VALUES

On success, **t_snd()** returns the number of bytes accepted by the transport provider. On failure, it returns –1 and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TFLOW	**O_NDELAY** was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N), **t_rcv**(3N)

Network Programming

NAME

t_snddis – send user-initiated disconnect request

SYNOPSIS

```
#include <tiuser.h>

int t_snddis(fd, call)
int fd;
struct t_call *call;
```

DESCRIPTION

t_snddis() is used to initiate an abortive release on an already established connection or to reject a connect request. *fd* identifies the local transport endpoint of the connection, and *call* specifies information associated with the abortive release. *call* points to a **t_call()** structure which is defined in **<nettlie/tiuser.h>** as:

```
struct t_call {
        struct netbuf addr;          /* address          */
        struct netbuf opt;           /* options          */
        struct netbuf udata;         /* user data        */
        int sequence;                /* sequence number  */
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). The values in *call* have different semantics, depending on the context of the call to **t_snddis()**. When rejecting a connect request, *call* must be non-NULL and contain a valid value of *sequence* to uniquely identify the rejected connect indication to the transport provider. The *addr* and *opt* fields of *call* are ignored. In all other cases, *call* need only be used when data is being sent with the disconnect request. The *addr*, *opt*, and *sequence* fields of the **t_call()** structure are ignored. If the user does not wish to send data to the remote user, the value of *call* may be NULL. *udata* specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider as returned by **t_open**(3N) or **t_getinfo**(3N). If the *len* field of *udata* is zero, no data will be sent to the remote user.

RETURN VALUES

t_snddis() returns:

0 on success.

−1 on failure and sets **t_errno** to indicate the error.

ERRORS

TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider's outgoing queue will be flushed, so data may be lost.
TBADF	The specified file descriptor does not refer to a transport endpoint.
TBADSEQ	An invalid sequence number was specified. The transport provider's outgoing queue will be flushed, so data may be lost.
	A NULL call structure was specified when rejecting a connect request. The transport provider's outgoing queue will be flushed, so data may be lost.
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The function was issued in the wrong sequence. The transport provider's outgoing queue may be flushed, so data may be lost.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_connect**(3N), **t_getinfo**(3N), **t_listen**(3N), **t_open**(3N)

Network Programming

NAME

t_sndrel – initiate an orderly release

SYNOPSIS

```
#include <tiuser.h>

int t_sndrel(fd)
int fd;
```

DESCRIPTION

t_sndrel() initiates an orderly release of a transport connection and indicates to the transport provider that the transport user has no more data to send. *fd* identifies the local transport endpoint where the connection exists. After issuing **t_sndrel()**, the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has been received.

t_sndrel() is an optional service of the transport provider, and is only supported if the transport provider returned service type **T_COTS_ORD** on **t_open**(3N) or **t_getinfo**(3N).

RETURN VALUES

t_sndrel() returns:

0	on success.
–1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TFLOW	**O_NDELAY** was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_open(3N), **t_rcvrel**(3N)

Network Programming

NAME

t_sndudata – send a data unit

SYNOPSIS

#include <tiuser.h>

```
int t_sndudata(fd, unitdata)
int fd;
struct t_unitdata *unitdata;
```

DESCRIPTION

t_sndudata( ) is used in connectionless mode to send a data unit to another transport user. *fd* identifies the local transport endpoint through which data will be sent, and *unitdata* points to a **t_unitdata** structure defined in **<nettli/tiuser.h>** as:

```
struct t_unitdata {
        struct netbuf addr;          /* address      */
        struct netbuf opt;           /* options      */
        struct netbuf udata;         /* user data    */
};
```

The *maxlen*, *len*, and *buf* members of the *netbuf* structure are described in **t_accept**(3N). In *unitdata*, *addr* specifies the protocol address of the destination user, *opt* identifies protocol-specific options that the user wants associated with this request, and *udata* specifies the user data to be sent. The user may choose not to specify what protocol options are associated with the transfer by setting the *len* field of *opt* to 0. In this case, the provider may use default options.

If the *len* field of *udata* is 0, no data unit will be passed to the transport provider; **t_sndudata**( ) will not send zero-length data units.

By default, **t_sndudata**( ) operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if **T_NDELAY** is set (using **t_open**(3N) or **fcntl**( )), **t_sndudata**( ) will execute in asynchronous mode and will fail under such conditions.

If **t_sndudata**( ) is issued from an invalid state, or if the amount of data specified in *udata* exceeds the TSDU size as returned by **t_open**( ) or **t_getinfo**(3N), the provider will generate an EPROTO protocol error. See TSYSERR below.

RETURN VALUES

t_sndudata( ) returns:

0	on success.
−1	on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TFLOW	**T_NDELAY** was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

intro(3), **t_rcvudata**(3N), **t_rcvuderr**(3N)

Network Programming

NAME

t_sync – synchronize transport library

SYNOPSIS

```
#include <tiuser.h>

int t_sync(fd)
int fd;
```

DESCRIPTION

For the transport endpoint specified by *fd*, **t_sync()** synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, it can convert a raw file descriptor (obtained using **open**(2V), **dup**(2V), or as a result of a **fork**(2V) and **execve**(2V)) to an initialized transport endpoint, assuming that file descriptor referenced a transport provider. **t_sync()** also allows two cooperating processes to synchronize their interaction with a transport provider.

For example, if a process *forks* a new process and issues an *exec*, the new process must issue a **t_sync()** to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activities so as not to violate the state of the provider. **t_sync()** returns the current state of the provider to the user, thereby enabling the user to verify the state before taking further action. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event could change the provider's state *after* a **t_sync()** is issued.

If the provider is undergoing a state transition when **t_sync()** is called, the function will fail.

RETURN VALUES

t_sync() returns −1 on failure. Upon success, the state of the transport provider is returned; it may be one of the following:

T_IDLE	idle
T_OUTCON	outgoing connection pending
T_INCON	incoming connection pending
T_DATAXFER	data transfer
T_OUTREL	outgoing orderly release (waiting for an orderly release indication)
T_INREL	incoming orderly release (waiting for an orderly release request)
T_UNBND	unbound

ERRORS

TBADF	The specified file descriptor is a valid open file descriptor but does not refer to a transport endpoint.
TSTATECHNG	The transport provider is undergoing a state change.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

dup(2V), **execve**(2V), **fork**(2V), **open**(2V)

Network Programming

NAME

t_unbind – disable a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_unbind(fd)
int fd;

DESCRIPTION

t_unbind() disables the transport endpoint specified by *fd* which was previously bound by **t_bind**(3N). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

RETURN VALUES

t_unbind() returns:

0 on success.

–1 on failure and sets **t_errno** to indicate the error.

ERRORS

TBADF	The specified file descriptor does not refer to a transport endpoint.
TLOOK	An asynchronous event has occurred on this transport endpoint.
TOUTSTATE	The function was issued in the wrong sequence.
TSYSERR	The function failed due to a system error and set **errno** to indicate the error.

SEE ALSO

t_bind(3N)

Network Programming

NAME

tcgetpgrp, tcsetpgrp – get, set foreground process group ID

SYNOPSIS

```
#include <sys/types.h>

pid_t tcgetpgrp(fd)
int fd;

int tcsetpgrp(fd, pgrp_id)
int fd;
pid_t pgrp_id;
```

DESCRIPTION

tcgetpgrp( ) returns the value of the process group ID of the foreground process group associated with the terminal (see NOTES). **tcgetpgrp( )** is allowed from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.

If the process has a controlling terminal, **tcsetpgrp( )** sets the foreground process group ID associated with the terminal to *pgrp_id*. The file associated with *fd* must be the controlling terminal and must be currently associated with the session of the calling process. The value of *pgrp_id* must match a process group ID of a process in the same session as the calling process.

RETURN VALUES

On success, **tcgetpgrp( )** returns the process group ID of the foreground process group associated with the terminal. On failure, it returns −1 and sets **errno** to indicate the error.

tcsetpgrp( ) returns:

0	on success.
−1	on failure and sets **errno** to indicate the error.

ERRORS

If any of the following conditions occur, **tcgetpgrp( )** sets **errno** to:

EBADF	*fd* is not a valid file descriptor.
ENOSYS	**tcgetpgrp( )** is not supported in this implementation.
ENOTTY	The calling process does not have a controlling terminal.
	The file is not the controlling terminal.

If any of the following conditions occur, **tcsetpgrp( )** sets **errno** to:

EBADF	*fd* is not a valid file descriptor.
EINVAL	The value of *pgrp_id* is not a valid process group ID.
ENOTTY	The calling process does not have a controlling terminal.
	The file is not the controlling terminal.
	The controlling terminal is no longer associated with the session of the calling process.
EPERM	The value of *pgrp_id* is a valid process group ID, but does not match the process group ID of a process in the same session as the calling process.

SEE ALSO

setpgid(2V), **setsid**(2V)

NOTES

For **tcgetpgrp()** and **tcsetpgrp()** to behave as described above, {_POSIX_JOB_CONTROL} must be in effect (see **sysconf**(2V)). {_POSIX_JOB_CONTROL} is always in effect on SunOS systems, but for portability, applications should call **sysconf()** to determine whether {_POSIX_JOB_CONTROL} is in effect for the current system.

If {_POSIX_JOB_CONTROL} is not defined on a system conforming to *IEEE Std 1003.1-1988* either **tcgetpgrp()** and **tcsetpgrp()** behave as described above, or **tcgetpgrp()** and **tcsetpgrp()** fail.

NAME

termcap, tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum (id)
char *id;

tgetflag (id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base **termcap**(5). These are low level routines; see **curses**(3V) for a higher level package.

tgetent( ) extracts the entry for terminal *name* into the *bp* buffer, with the current size of the tty (usually a window). This allows pre-SunWindows programs to run in a window of arbitrary size. *bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to **tgetnum**( ), **tgetflag**( ), and **tgetstr**( ). **tgetent**( ) returns −1 if it cannot open the **termcap**( ) file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a **TERMCAP** variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the environment string **TERM**, the **TERMCAP** string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than **/etc/termcap**. This can speed up entry into programs that call **tgetent**, as well as to help debug new terminal descriptions or to make one for your terminal if you cannot write the file **/etc/termcap**. Note: if the window size changes, the "lines" and "columns" entries in *bp* are no longer correct. See the *SunView Programmer's Guide* for details regarding [how to handle] this.

tgetnum( ) gets the numeric value of capability **ID**, returning −1 if is not given for the terminal. **tgetflag**( ) returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. **tgetstr**( ) gets the string value of capability **ID**, placing it in the buffer at *area*, advancing the *area* pointer. It decodes the abbreviations for this field described in **termcap**(5), except for cursor addressing and padding information. **tgetstr**( ) returns the string pointer if successful. Otherwise it returns zero.

tgoto() returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline*. It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **\n**, **^D** or **^@** in the returned string. (Programs which call **tgoto()** should be sure to turn off the **XTABS bit(s),since tgoto()** may now output a tab. Note: programs using **termcap()** should in general turn off **XTABS** anyway since some terminals use **^I** (CTRL-I) for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then **tgoto()** returns **OOPS**.

tputs() decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable, *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the encoded output speed of the terminal as described in **tty**(4). The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a NULL (^@) is inappropriate.

FILES

/usr/lib/libtermcap.a	–ltermcap library
/etc/termcap	data base

SEE ALSO

ex(1), **curses**(3V), **tty**(4), **termcap**(5)

NAME

termios, tcgetattr, tcsetattr, tcsendbreak, tcdrain, tcflush, tcflow, cfgetospeed, cfgetispeed, cfsetispeed, cfsetospeed – get and set terminal attributes, line control, get and set baud rate, get and set terminal foreground process group ID

SYNOPSIS

```
#include <termios.h>
#include <unistd.h>

int tcgetattr(fd, termios_p)
int fd;
struct termios *termios_p;

int tcsetattr(fd, optional_actions, termios_p)
int fd;
int optional_actions;
struct termios *termios_p;

int tcsendbreak(fd, duration)
int fd;
int duration;

int tcdrain(fd)
int fd;

int tcflush(fd, queue_selector)
int fd;
int queue_selector;

int tcflow(fd, action)
int fd;
int action;

speed_t cfgetospeed(termios_p)
struct termios *termios_p;

int cfsetospeed(termios_p, speed)
struct termios *termios_p;
speed_t speed;

speed_t cfgetispeed(termios_p)
struct termios *termios_p;

int cfsetispeed(termios_p, speed)
struct termios *termios_p;
speed_t speed;

#include <sys/types.h>
#include <termios.h>
```

DESCRIPTION

The termios functions describe a general terminal interface that is provided to control asynchronous communications ports. A more detailed overview of the terminal interface can be found in **termio**(4). That section also describes an **ioctl**( ) interface that can be used to access the same functionality. However, the function interface described here is the preferred user interface.

Many of the functions described here have a *termios_p* argument that is a pointer to a **termios** structure. This structure contains the following members:

```
tcflag_t    c_iflag;        /* input modes */
tcflag_t    c_oflag;        /* output modes */
tcflag_t    c_cflag;        /* control modes */
tcflag_t    c_lflag;        /* local modes */
cc_t        c_cc[NCCS];     /* control chars */
```

These structure members are described in detail in **termio**(4).

tcgetattr( ) gets the parameters associated with the object referred by *fd* and stores them in the **termios** structure referenced by *termios_p*. This function may be invoked from a background process; however, the terminal attributes may be subsequently changed by a foreground process.

tcsetattr( ) sets the parameters associated with the terminal (unless support is required from the underlying hardware that is not available) from the **termios** structure referred to by *termios_p* as follows:

- If *optional_actions* is **TCSANOW**, the change occurs immediately.
- If *optional_actions* is **TCSADRAIN**, the change occurs after all output written to *fd* has been transmitted. This function should be used when changing parameters that affect output.
- If *optional_actions* is **TCSAFLUSH**, the change occurs after all output written to the object referred by *fd* has been transmitted, and all input that has been received but not read will be discarded before the change is made.

The symbolic constants for the values of *optional_actions* are defined in **<sys/termios.h>**.

If the terminal is using asynchronous serial data transmission, **tcsendbreak**( ) transmits a continuous stream of zero-valued bits for a specific duration. If *duration* is zero, it transmits zero-valued bits for at least 0.25 seconds, and not more that 0.5 seconds. If *duration* is not zero, it sends zero-valued bits for *duration***N* seconds, where *N* is at least 0.25, and not more than 0.5.

If the terminal is not using asynchronous serial data transmission, **tcsendbreak**( ) returns without taking any action.

tcdrain( ) waits until all output written to the object referred to by *fd* has been transmitted.

tcflush( ) discards data written to the object referred to by *fd* but not transmitted, or data received but not read, depending on the value of *queue_selector*:

- If *queue_selector* is **TCIFLUSH**, it flushes data received but not read.
- If *queue_selector* is **TCOFLUSH**, it flushes data written but not transmitted.
- If *queue_selector* is **TCIOFLUSH**, it flushes both data received but not read, and data written but not transmitted.

The symbolic constants for the values of *queue_selector* and *action* are defined in **termios.h**.

The default on open of a terminal file is that neither its input nor its output is suspended.

tcflow( ) suspends transmission or reception of data on the object referred to by *fd*, depending on the value of *actions*:

- If action is **TCOOFF**, it suspends output.
- If action is **TCOON**, it restarts suspended output.
- If action is **TCIOFF**, the system transmits a STOP character, which stops the terminal device from transmitting data to the system. (See **termio**(4).)
- If action is **TCION**, the system transmits a START character, which starts the terminal device transmitting data to the system. (See **termio**(4).)

The baud rate functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure. The effects on the terminal device described below do not become effective until **tcsetattr**( ) is successfully called.

The input and output baud rates are stored in the **termios** structure. The values shown in the table are supported. The names in this table are defined in **termios.h**

Name	Description	Name	Description
B0	Hang up	**B600**	600 baud
B50	50 baud	**B1200**	1200 baud
B75	75 baud	**B1800**	1800 baud
B110	110 baud	**B2400**	2400 baud
B134	134.5 baud	**B4800**	4800 baud
B150	150 baud	**B9600**	9600 baud
B200	200 baud	**B19200**	19200 baud
B300	300 baud	**B38400**	38400 baud

cfgetospeed( ) returns the output baud rate stored in the **termios** structure pointed to by *termios_p*.

cfsetospeed( ) sets the output baud rate stored in the **termios** structure pointed to by *termios_p* to *speed*. The zero baud rate, **B0**, is used to terminate the connection. If **B0** is specified, the modem control lines shall no longer be asserted. Normally, this will disconnect the line.

If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate.

cfgetispeed( ) returns the input baud rate stored in the **termios** structure.

cfsetispeed( ) sets the input baud rate stored in the **termios** structure to *speed*.

RETURN VALUES

cfgetispeed( ) returns the input baud rate stored in the **termios** structure.

cfgetospeed( ) returns the output baud rate stored in the **termios** structure.

cfsetispeed( ) and **cfsetospeed( )** return:

0 on success.

−1 on failure and sets **errno** to indicate the error.

All other functions return:

0 on success.

−1 on failure and set **errno** to indicate the error.

ERRORS

EBADF The *fd* argument is not a valid file descriptor.

ENOTTY The file associated with *fd* is not a terminal.

tcsetattr( ) may set **errno** to:

EINVAL The *optional_actions* argument is not a proper value.

An attempt was made to change an attribute represented in the **termios** structure to an unsupported value.

tcsendbreak( ) may set **errno** to:

EINVAL The device does not support **tcsendbreak( )**.

tcdrain( ) may set **errno** to:

EINTR A signal interrupted **tcdrain( )**.

EINVAL The device does not support **tcdrain( )**.

tcflush( ) may set **errno** to:

EINVAL The device does not support **tcflush( )**.

The *queue_selector* argument is not a proper value.

tcflow( ) may set **errno** to:

EINVAL The device does not support **tcflow( )**.

The *action* argument is not a proper value.

tcsetattr( ) may set **errno** to:

EAGAIN There is insufficient memory available to copy in the arguments.

EBADF *fd* is not a valid descriptor.

EFAULT Some part of the structure pointed to by *termios_p* is outside the process's allocated address space.

EINVAL *optional_actions* is not valid.

EIO The calling process is a background process.

ENOTTY *fd* does not refer to a terminal device.

ENXIO The terminal referred to by *fd* is hung up.

cfsetispeed( ) and **cfsetospeed( )** may set **errno** to:

EINVAL *speed* is greater than B38400 or less than 0.

SEE ALSO

setpgid(2V), **setsid**(2V), **termio**(4)

NAME

time, ftime – get date and time

SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>

time_t time(tloc)
time_t *tloc;

#include <sys/timeb.h>

int ftime(tp)
struct timeb *tp;
```

DESCRIPTION

time() returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is non-NULL, the return value is also stored in the location to which *tloc* points.

ftime() fills in a structure pointed to by *tp*, as defined in **<sys/timeb.h>**:

```
struct timeb
{
        time_t   time;
        unsigned short millitm;
        short    timezone;
        short    dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

RETURN VALUES

time() returns the value of time on success. On failure, it returns **(time_t)** –1.

On success, **ftime()** returns no useful value. On failure, it returns –1.

SEE ALSO

date(1V), **gettimeofday**(2), **ctime**(3V)

NAME

times – get process times

SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

int times(buffer)
struct tms *buffer;
```

SYSTEM V SYNOPSIS

```
clock_t times(buffer)
struct tms *buffer;
```

DESCRIPTION

This interface is obsoleted by **getrusage**(2).

times( ) returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

buffer points to the following structure:

```
struct tms {
        clock_t tms_utime;          /* user time */
        clock_t tms_stime;          /* system time */
        clock_t tms_cutime;         /* user time, children */
        clock_t tms_cstime;         /* system time, children */
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a **wait**(2V).

tms_utime is the CPU time used while executing instructions in the user space of the calling process.

tms_stime is the CPU time used by the system on behalf of the calling process.

tms_cutime is the sum of the **tms_utime**s and **tms_cutime**s of the child processes.

tms_cstime is the sum of the **tms_stime**s and **tms_cstime**s of the child processes.

RETURN VALUES

times( ) returns:

0 on success.

–1 on failure.

SYSTEM V RETURN VALUES

Upon successful completion, **times**( ) returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past. This point does not change from one invocation of **times**( ) to another within the same process. On failure, **times**( ) returns **(clock_t)** –1.

SEE ALSO

time(1V), **getrusage**(2), **wait**(2V), **time**(3V)

NAME

timezone – get time zone name given offset from GMT

SYNOPSIS

char *timezone(zone, dst)

DESCRIPTION

timezone( ) attempts to return the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Savings Time version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; for instance, in Afghanistan '**timezone(–(60*4+30), 0)**' is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

Note: the offset westward from Greenwich and an indication of whether Daylight Savings Time is in effect may not be sufficient to determine the name of the time zone, as the name may differ between different locations in the same time zone. Instead of using **timezone( )** to determine the name of the time zone for a given time, that time should be converted to a '**struct tm**' using **localtime( )** (see **ctime**(3V)) and the *tm_zone* field of that structure should be used. **timezone( )** is retained for compatibility with existing programs.

SEE ALSO

ctime(3V)

NAME

tmpfile – create a temporary file

SYNOPSIS

#include <stdio.h>

FILE *tmpfile( )

DESCRIPTION

tmpfile( ) creates a temporary file using a name generated by **tmpnam**(3S), and returns a corresponding **FILE** pointer. If the file cannot be opened, an error message is printed using **perror**(3), and a NULL pointer is returned. The file will automatically be deleted when the process using it terminates. The file is opened for update (**"w+"**).

SEE ALSO

creat(2V), **unlink**(2V), **fopen**(3V), **mktemp**(3), **perror**(3), **tmpnam**(3S)

NAME

tmpnam, tempnam – create a name for a temporary file

SYNOPSIS

#include <stdio.h>

char *tmpnam (s)
char *s;

char *tempnam (dir, pfx)
char *dir, *pfx;

DESCRIPTION

These functions generate file names that can safely be used for a temporary file.

tmpnam() always generates a file name using the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file. If *s* is NULL, **tmpnam()** leaves its result in an internal static area and returns a pointer to that area. The next call to **tmpnam()** will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least **L_tmpnam** bytes, where **L_tmpnam** is a constant defined in **<stdio.h>**; **tmpnam()** places its result in that array and returns *s*.

tempnam() allows the user to control the choice of a directory. The argument **dir** points to the name of the directory in which the file is to be created. If **dir** is NULL or points to a string which is not a name for an appropriate directory, the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file is used. If that directory is not accessible, **/tmp** will be used as a last resort. This entire sequence can be up-staged by providing an environment variable **TMPDIR** in the user's environment, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

tempnam() uses **malloc()** to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from **tempnam()** may serve as an argument to *free* (see **malloc**(3V)). If **tempnam()** cannot return the expected result for any reason, that is, **malloc()** failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.

NOTES

These functions generate a different file name each time they are called.

Files created using these functions and either **fopen()** or **creat()** are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use **unlink**(2V) to remove the file when its use is ended.

SEE ALSO

creat(2V), **unlink**(2V), **fopen**(3V), **malloc**(3V), **mktemp**(3), **tmpfile**(3S)

BUGS

If called more than 17,576 times in a single process, these functions will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or **mktemp()**, and the file names are chosen so as to render duplication by other means unlikely.

NAME

tsearch, tfind, tdelete, twalk – manage binary search trees

SYNOPSIS

#include <search.h>

char *tsearch((char *) key, (char **) rootp, compar)
int (*compar)( );

char *tfind((char *) key, (char **) rootp, compar)
int (*compar)( );

char *tdelete((char *) key, (char **) rootp, compar)
int (*compar)( );

void twalk((char *) root, action)
void (*action)( );

DESCRIPTION

tsearch(), **tfind()**, **tdelete()**, and **twalk()** are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

tsearch() is used to build and access the tree. *key* is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to **key* (the value pointed to by *key*), a pointer to this found datum is returned. Otherwise, **key* is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. *rootp* points to a variable that points to the root of the tree. A NULL value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like **tsearch()**, **tfind()** will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, **tfind()** will return a NULL pointer. The arguments for **tfind()** are the same as for **tsearch()**.

tdelete() deletes a node from a binary search tree. The arguments are the same as for **tsearch()**. The variable pointed to by *rootp* will be changed if the deleted node was the root of the tree. **tdelete()** returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

twalk() traverses a binary search tree. *root* is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type **typedef enum { preorder, postorder, endorder, leaf }** VISIT; (defined in the **<search.h>** header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

void twalk();
char *tsearch();

struct node {           /* pointers to these are stored in the tree */
        char *string;
        int count;
};

#define MAXNODES        12
#define MAXSTRING       100
#define MINSTRING       3               /* char, newline, eos */

char string_space[MAXSTRING];           /* space to store strings */
struct node node_space[MAXNODES];       /* nodes to store */
struct node *root = NULL;               /* this points to the root */

main()
{
        char *strptr = string_space;
        int maxstrlen = MAXSTRING;
        struct node *nodeptr = node_space;
        int node_compare();
        void print_node();
        struct node **found;
        int length;

        while (fgets(strptr, maxstrlen, stdin) != NULL) {
                /* remove the trailing newline */
                length = strlen(strptr);
                strptr[length-1] = 0;
                /* set node */
                nodeptr->string = strptr;
                /* locate node into the tree */
                found = (struct node **)
                    tsearch((char *) nodeptr, (char **) &root, node_compare);
                /* bump the count */
                (*found)->count++;

                if (*found == nodeptr) {
                        /* node was inserted, so get a new one */
                        strptr += length;
                        maxstrlen -= length;
                        if (maxstrlen < MINSTRING)
                                break;
                        if (++nodeptr >= &node_space[MAXNODES])
                                break;
                }
        }
        twalk((char *)root, print_node);
}
```

```
/*
    This routine compares two nodes, based on an
    alphabetical ordering of the string field.
*/

int node_compare(node1, node2)
        struct node *node1, *node2;
{
        return strcmp(node1->string, node2->string);
}

/* Print out nodes in alphabetical order */
/*ARGSUSED2*/
void
print_node(node, order, level)
        struct node **node;
        VISIT order;
        int level;
{
        if (order == postorder || order == leaf) {
                (void) printf("string = %20s,  count = %d0,
                   (*node)->string, (*node)->count);
        }
}
```

SEE ALSO

bsearch(3), **hsearch**(3), **lsearch**(3)

DIAGNOSTICS

A NULL pointer is returned by **tsearch**( ) if there is not enough space available to create a new node.

A NULL pointer is returned by **tsearch**( ), **tfind**( ) and **tdelete**( ) if *rootp* is NULL on entry.

If the datum is found, both **tsearch**( ) and **tfind**( ) return a pointer to it. If not, **tfind**( ) returns NULL, and **tsearch**( ) returns a pointer to the inserted item.

WARNINGS

The *root* argument to **twalk**( ) is one level of indirection less than the *rootp* arguments to **tsearch**( ) and **tdelete**( ).

There are two nomenclatures used to refer to the order in which tree nodes are visited. **tsearch**( ) uses preorder, postorder and endorder to respectively refer to visting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

BUGS

If the calling function alters the pointer to the root, results are unpredictable.

NAME

ttyname, isatty – find name of a terminal

SYNOPSIS

char *ttyname(fd)
int fd;

int isatty(fd)
int fd;

DESCRIPTION

ttyname( ) returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fd*.

isatty( ) returns 1 if *fd* is associated with a terminal device, 0 otherwise.

FILES

/dev/*

SEE ALSO

ctermid(3V), **ioctl**(2), **ttytab**(5)

RETURN VALUES

On success, **ttyname**( ) returns a pointer to the terminal device. If *fd* does not describe a terminal device in directory **/dev**, **ttyname**( ) returns NULL.

isatty( ) returns 1 if *fd* is associated with a terminal device. It returns 0 otherwise.

BUGS

The return value points to static data which are overwritten by each call.

NAME

ttyslot – find the slot in the utmp file of the current process

SYNOPSIS

int ttyslot()

DESCRIPTION

ttyslot() returns the index of the current user's entry in **/etc/utmp**. This is accomplished by actually scanning the file **/etc/ttytab** for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

RETURN VALUES

On success, **ttyslot()** returns the index of the current user's entry in **/etc/utmp**. If an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device, **ttyslot()** returns 0.

SYSTEM V RETURN VALUES

If an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device, **ttyslot()** returns −1.

FILES

/etc/ttytab
/etc/utmp

NAME

ualarm – schedule signal after interval in microseconds

SYNOPSIS

unsigned ualarm(value, interval)
unsigned value;
unsigned interval;

DESCRIPTION

This is a simplified interface to setitimer( ) (see **getitimer**(2)).

ualarm( ) sends signal **SIGALRM**, see **signal**(3V), to the invoking process in a number of microseconds given by the *value* argument. Unless caught or ignored, the signal terminates the process.

If the *interval* argument is non-zero, the **SIGALRM** signal will be sent to the process every *interval* microseconds after the timer expires (for instance, after *value* microseconds have passed).

Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 microseconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

getitimer(2), **sigpause**(2V), **sigvec**(2), **alarm**(3V), **signal**(3V), **sleep**(3V), **usleep**(3)

NAME

ulimit – get and set user limits

SYNOPSIS

long ulimit(cmd, newlimit)
int cmd;
long newlimit;

DESCRIPTION

This function is included for System V compatibility.

This routine provides for control over process limits. The *cmd* values available are:

1 Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2 Set the process's file size limit to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. **ulimit()** will fail and the limit will be unchanged if a process with an effective user ID other than the super-user attempts to increase its file size limit.

3 Get the maximum possible break value. See **brk**(2).

4 Get the size of the process' file descriptor table, as returned by **getdtablesize**(2).

RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise a value of –1 is returned and **errno** is set to indicate the error.

ERRORS

EPERM A user other than the super-user attempted to increase the file size limit.

SEE ALSO

brk(2), **getdtablesize**(2), **getrlimit**(2), **write**(2V)

NAME

ungetc – push character back into input stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
```

DESCRIPTION

ungetc( ) pushes the character *c* back onto an input stream. That character will be returned by the next **getc**( ) call on that stream. **ungetc**( ) returns *c*, and leaves the file stream unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that stream is **stdin**, one character may be pushed back onto the buffer without a previous read statement.

If *c* equals EOF, **ungetc**( ) does nothing to the buffer and returns EOF.

An **fseek**(3S) erases all memory of pushed back characters.

SEE ALSO

fseek(3S), **getc**(3V), **setbuf**(3V)

DIAGNOSTICS

ungetc( ) returns EOF if it cannot push a character back.

NAME

usleep – suspend execution for interval in microseconds

SYNOPSIS

usleep(useconds)
unsigned useconds;

DESCRIPTION

Suspend the current process for the number of microseconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system, or because of the time spent in processing the call.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent a short time later.

This routine is implemented using **setitimer()** (see **getitimer**(2)); it requires eight system calls each time it is invoked. A similar but less compatible function can be obtained with a single **select**(2); it would not restart after signals, but would not interfere with other uses of **setitimer**.

SEE ALSO

getitimer(2), **sigpause**(2V), **alarm**(3V), **sleep**(3V), **ualarm**(3)

NAME

utime – set file times

SYNOPSIS

```
#include <utime.h>

int utime(path, times)
char *path;
struct utimbuf *times;
```

DESCRIPTION

utime( ) sets the access and modification times of the file named by *path*.

If *times* is NULL, the access and modification times are set to the current time. The effective user ID (UID) of the calling process must match the owner of the file or the process must have write permission for the file to use **utime**( ) in this manner.

If *times* is not NULL, it is assumed to point to a **utimbuf** structure, defined in **<utime.h>** as:

```
struct utimbuf {
        time_t  actime;    /* set the access time */
        time_t  modtime; /* set the modification time */
};
```

The access time is set to the value of the first member, and the modification time is set to the value of the second member. The times contained in this structure are measured in seconds since 00:00:00 GMT Jan 1, 1970. Only the owner of the file or the super-user may use **utime**( ) in this manner.

Upon successful completion, **utime**( ) marks for update the *st_ctime* field of the file.

RETURN VALUES

utime( ) returns:

0 on success.

–1 on failure and sets **errno** to indicate the error.

ERRORS

EACCES	Search permission is denied for a component of the path prefix of *path*.
EACCES	The effective user ID is not super-user and not the owner of the file, write permission is denied for the file, and *times* is NULL.
EFAULT	*path* or *times* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating *path*.
ENAMETOOLONG	The length of *path* exceeds {PATH_MAX}.
	A pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect (see **pathconf**(2V)).
ENOENT	The file referred to by *path* does not exist.
ENOTDIR	A component of the path prefix of *path* is not a directory.
EPERM	The effective user ID of the process is not super-user and not the owner of the file, and *times* is not NULL.
EROFS	The file system containing the file is mounted read-only.

SYSTEM V ERRORS

In addition to the above, the following may also occur:

ENOENT	*path* points to an empty string.

SEE ALSO

pathconf(2V), **stat**(2V), **utimes**(2)

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (for instance, int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT	
LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE	
LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT	
LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE	
LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3), **intro**(3M)

NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

function(va_alist) va_dcl

va_list pvar;

va_start(pvar);

f = va_arg(pvar, type);

va_end(pvar);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as **printf**(3V)) but do not use **varargs**( ) are inherently nonportable, since different machines use different argument passing conventions. Routines with variable arguments lists *must* use **varargs**( ) functions in order to run correctly on Sun-4 systems.

va_alist( ) is used in a function header to declare a variable argument list.

va_dcl( ) is a declaration for **va_alist**( ). No semicolon should follow **va_dcl**( ).

va_list( ) is a type defined for the variable used to traverse the list. One such variable must always be declared.

va_start(*pvar*) is called to initialize *pvar* to the beginning of the list.

va_arg(*pvar*, *type*) will return the next argument in the list pointed to by *pvar*. The parameter *type* is a type name such that the type of a pointer to an object that has the specified type can be obtained simply by appending a * to *type*. If *type* disagrees with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.

In standard C, arguments that are **char** or **short** are converted to **int** and should be accessed as **int**, arguments that are **unsigned char** or **unsigned short** are converted to **unsigned int** and should be accessed as **unsigned int**, and arguments that are **float** are converted to **double** and should be accessed as **double**. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

va_end(*pvar*) is used to finish up.

Multiple traversals, each bracketed by **va_start**( ) ... **va_end**( ), are possible.

va_alist( ) must encompass the entire arguments list. This insures that a **#define** statement can be used to redefine or expand its value.

The argument list (or its remainder) can be passed to another function using a pointer to a variable of type **va_list**( ) — in which case a call to **va_arg**( ) in the subroutine advances the argument-list pointer with respect to the caller as well.

EXAMPLE

This example is a possible implementation of **execl**(3V).

```
#include <varargs.h>
#define MAXARGS     100

/*      execl is called by
 *      execl(file, arg1, arg2, ..., (char *)0);
 */
execl (va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];
        int argno = 0;

        va_start (ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
                ;
        va_end (ap);
        return execv(file, args);
}
```

SEE ALSO

execl(3V), **printf**(3V)

BUGS

It is up to the calling routine to specify how many arguments there are, since it is not possible to determine this from the stack frame. For example, **execl**( ) is passed a zero pointer to signal the end of the list. **printf**( ) can tell how many arguments are supposed to be there by the format.

The macros **va_start**( ) and **va_end**( ) may be arbitrarily complex; for example, **va_start**( ) might contain an opening brace, which is closed by a matching brace in **va_end**( ). Thus, they should only be used where they could be placed within a single complex statement.

NAME

vlimit – control maximum system resource consumption

SYNOPSIS

#include <sys/vlimit.h>

vlimit(resource, value) int resource, value;

DESCRIPTION

This facility is superseded by getrlimit(2).

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified resource. If *value* is specified as −1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

LIM_NORAISE	A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction.
LIM_CPU	the maximum number of CPU-seconds to be used by each process
LIM_FSIZE	the largest single file which can be created
LIM_DATA	the maximum growth of the data+stack region using **sbrk**() (see **brk**(2)) beyond the end of the program text
LIM_STACK	the maximum size of the automatically-extended stack region
LIM_CORE	the size of the largest core dump that will be created.
LIM_MAXRSS	a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared **LIM_MAXRSS.**

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to **csh**(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file I/O operation which would create a file which is too large will cause a signal **SIGXFSZ** to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal **SIGXCPU** is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the CPU time limit.

SEE ALSO

csh(1), **sh**(1), **brk**(2)

BUGS

If **LIM_NORAISE** is set, then no grace should be given when the CPU time limit is exceeded.

There should be *limit* and *unlimit* commands in **sh**(1) as well as in **csh**(1).

NAME

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf(format, ap)
char *format;
va_list ap;

int vfprintf(stream, format, ap)
FILE *stream;
char *format;
va_list ap;

char *vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

SYSTEM V SYNOPSIS

```
int vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

vprintf( ), **vfprintf**( ), and **vsprintf**( ) are the same as **printf**(3V), **fprintf**( ), and **sprintf**( ) (see **printf**(3V)) respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **varargs**(3).

RETURN VALUES

On success, **vprintf**( ) and **vfprintf**( ) return the number of characters transmitted, excluding the null character. On failure, they return EOF.

vsprintf( ) returns *s*.

SYSTEM V RETURN VALUES

vsprintf( ) returns the number of characters transmitted, excluding the null character.

EXAMPLES

The following demonstrates how **vfprintf**( ) could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
...
        /* error should be called like:
         *       error(function_name, format, arg1, arg2...);
         * Note: function_name and format cannot be declared
         * separately because of the definition of varargs.
         */

/*VARARGS0*/
void
error (va_alist)
        va_dcl
{
        va_list args;
        char *fmt;

        va_start(args);
                /* print name of function causing error */
```

```
                (void) fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
                fmt = va_arg(args, char *);
                        /* print out remainder of message */
                (void) vfprintf(stderr, fmt, args);
                va_end(args);
                (void) abort();
        }
```

SEE ALSO

printf(3V), **varargs**(3)

NAME

vsyslog – log message with a varargs argument list

SYNOPSIS

```
#include <syslog.h>
#include <varargs.h>

int vsyslog(priority, message, ap)
char *message;
va_list ap;
```

DESCRIPTION

vsyslog() is the same as **syslog(3)** except that instead of being called with a variable number of arguments, it is called with an argument list as defined by **varargs(3)**.

EXAMPLE

The following demonstrates how **vsyslog()** could be used to write an error routine.

```
#include <syslog.h>
#include <varargs.h>
...
        /* error should be called like:
         *       error(pri, function_name, format, arg1, arg2...);
         * Note that pri, function_name, and format cannot be declared
         * separately because of the definition of varargs.
         */

/*VARARGS0*/
void
error(va_alist)
        va_dcl;
{
        va_list args;
        int pri;
        char *message;

        va_start(args);
        pri = va_arg(args, int);
                /* log name of function causing error */
        (void) syslog(pri, "ERROR in %s", va_arg(args, char *));
        message = va_arg(args, char *);
                /* log remainder of message */
        (void) vsyslog(pri, fmt, args);
        va_end(args);
        (void) abort();
}
```

SEE ALSO

syslog(3), **varargs(3)**

NAME

vtimes – get information about resource utilization

SYNOPSIS

```
vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;
```

DESCRIPTION

Note: this facility is superseded by **getrusage**(2).

vtimes( ) returns accounting information for the current process and for the terminated child processes of the current process. Either *par_vm* or *ch_vm* or both may be 0, in which case only the information for the pointers which are non-zero is returned.

After the call, each buffer contains information as defined by the contents of the include file **<sys/vtimes.h>**:

```
struct vtimes {
        int      vm_utime;          /* user time (*HZ) */
        int      vm_stime;          /* system time (*HZ) */
        /* divide next two by utime+stime to get averages */
        unsigned vm_idsrss;         /* integral of d+s rss */
        unsigned vm_ixrss;          /* integral of text rss */
        int      vm_maxrss;         /* maximum rss */
        int      vm_majflt;         /* major page faults */
        int      vm_minflt;         /* minor page faults */
        int      vm_nswap;          /* number of swaps */
        int      vm_inblk;          /* block reads */
        int      vm_oublk;          /* block writes */
};
```

The **vm_utime** and **vm_stime** fields give the user and system time respectively in 60ths of a second (or 50ths if that is the frequency of wall current in your locality.) The **vm_idrss** and **vm_ixrss** measure memory usage. They are computed by integrating the number of memory pages in use each over cpu time. They are reported as though computed discretely, adding the current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core pages over 1 cpu-second for its data and stack, then **vm_idsrss** would have the value 5*60, where **vm_utime+vm_stime** would be the 60. **vm_idsrss** integrates data and stack segment usage, while **vm_ixrss** integrates text segment usage. **vm_maxrss** reports the maximum instantaneous sum of the text+data+stack core-resident page count.

The **vm_majflt** field gives the number of page faults which resulted in disk activity; the **vm_minflt** field gives the number of page faults incurred in simulation of reference bits; **vm_nswap** is the number of swaps which occurred. The number of file system input/output events are reported in **vm_inblk** and **vm_oublk** These numbers account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

SEE ALSO

getrusage(2), **wait**(2V)

NAME

xdr – library routines for external data representation

SYNOPSIS AND DESCRIPTION

XDR routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls (RPC) are encoded and decoded using these routines. See **rpc**(3N).

All XDR routines require the header **<rpc/xdr.h>** to be included.

The XDR routines have been grouped by usage on the following man pages.

xdr_admin(3N) Library routines for managing the XDR stream. The routines documented on this page include:

- **xdr_getpos( )**
- **xdr_inline( )**
- **xdrrec_endofrecord( )**
- **xdrrec_eof( )**
- **xdrrec_readbytes( )**
- **xdrrec_skiprecord( )**
- **xdr_setpos( )**

xdr_complex(3N) Library routines for translating complex data types into their external data representation. The routines documented on this page include:

- **xdr_array( )**
- **xdr_bytes( )**
- **xdr_opaque( )**
- **xdr_pointer( )**
- **xdr_reference( )**
- **xdr_string( )**
- **xdr_union( )**
- **xdr_vector( )**
- **xdr_wrapstring( )**

xdr_create(3N) Library routines for creating XDR streams. The routines documented on this page include:

- **xdr_destroy( )**
- **xdrmem_create( )**
- **xdrrec_create( )**
- **xdrstdio_create( )**

xdr_simple(3N) Library routines for translating simple data types into their external data representation. The routines documented on this page include:

- **xdr_bool( )**
- **xdr_char( )**
- **xdr_double( )**
- **xdr_enum( )**
- **xdr_float( )**
- **xdr_free( )**
- **xdr_int( )**
- **xdr_long( )**
- **xdr_short( )**
- **xdr_u_char( )**
- **xdr_u_int( )**
- **xdr_u_long( )**
- **xdr_u_short( )**
- **xdr_void( )**

SEE ALSO

rpc(3N), **xdr_admin**(3N), **xdr_complex**(3N), **xdr_create**(3N), **xdr_simple**(3N)

Network Programming

NAME

xdr_getpos, xdr_inline, xdrrec_endofrecord, xdrrec_eof, xdrrec_readbytes, xdrrec_skiprecord, xdr_setpos – library routines for management of the XDR stream

DESCRIPTION

XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.

These routines deal specifically with the management of the XDR stream.

Routines

The **XDR** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/xdr.h>

u_int xdr_getpos(xdrs)
XDR *xdrs;

Invoke the get-position routine associated with the XDR stream, *xdrs*. The routine returns an unsigned integer, which indicates the position of the XDR byte stream. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances need not guarantee this.

long * xdr_inline(xdrs, len)
XDR *xdrs;
int len;

Invoke the in-line routine associated with the XDR stream, *xdrs*. The routine returns a pointer to a contiguous piece of the stream's buffer; *len* is the byte length of the desired buffer. Note: A pointer is cast to **long ***.

Warning: **xdr_inline()** may return NULL if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency.

bool_t xdrrec_endofrecord(xdrs, sendnow)
XDR *xdrs;
int sendnow;

This routine can be invoked only on streams created by **xdrrec_create()** (see **xdr_create**(3N)). The data in the output buffer is marked as a completed record, and the output buffer is optionally written out if *sendnow* is non-zero. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdrrec_eof(xdrs)
XDR *xdrs;
int empty;

This routine can be invoked only on streams created by **xdrrec_create()** (see **xdr_create**(3N)). After consuming the rest of the current record in the stream, this routine returns TRUE if the stream has no more input, FALSE otherwise.

int xdrrec_readbytes(xdrs, addr, nbytes)
XDR *xdrs;
caddr_t addr;
u_int nbytes;

This routine can be invoked only on streams created by **xdrrec_create()** (see **xdr_create**(3N)). It attempts to read *nbytes* bytes from the XDR stream into the buffer pointed to by *addr*. On success it returns the number of bytes read. Returns –1 on failure. A return value of 0 indicates an end of record.

bool_t xdrrec_skiprecord(xdrs)
XDR *xdrs;

This routine can be invoked only on streams created by **xdrrec_create**( ) (see **xdr_create**(3N)). It tells the XDR implementation that the rest of the current record in the stream's input buffer should be discarded. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_setpos(xdrs, pos)
XDR *xdrs;
u_int pos;

Invoke the set position routine associated with the XDR stream *xdrs*. The parameter *pos* is a position value obtained from **xdr_getpos( )**. This routine returns 1 if the XDR stream could be repositioned, and 0 otherwise.

Warning: It is difficult to reposition some types of XDR streams, so this routine may fail with one type of stream and succeed with another.

SEE ALSO

xdr(3N), **xdr_complex**(3N), **xdr_create**(3N), **xdr_simple**(3N)

NAME

xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring – library routines for translating complex data types

DESCRIPTION

XDR library routines allow C programmers to describe complex data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.

Routines

The **XDR** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/xdr.h>

```
bool_t xdr_array(xdrs, arrp, sizep, maxsize, elsize, elproc)
XDR *xdrs;
char **arrp;
u_int *sizep, maxsize, elsize;
xdrproc_t elproc;
```

A filter primitive that translates between a variable-length array and its corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *sizep* is the address of the element count of the array. This value is used by the filter while encoding and is set by it while decoding; the routine fails if the element count exceeds *maxsize*. The parameter *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

```
bool_t xdr_bytes(xdrs, arrp, sizep, maxsize)
XDR *xdrs;
char **arrp;
u_int *sizep, maxsize;
```

A filter primitive that translates between an array of bytes and its external representation. It treats the array of bytes as opaque data. The parameter *arrp* is the address of the array of bytes. While decoding if **arrp* is NULL, then the necessary storage is allocated to hold the array. This storage can be freed by using **xdr_free()** (see **xdr_simple**(3N)). *sizep* is the pointer to the actual length specifier for the array. This value is used by the filter while encoding and is set by it when decoding. *maxsize* is the maximum length of the array. The routine fails if the actual length of the array is greater than *maxsize* This routine returns TRUE if it succeeds, FALSE otherwise.

```
bool_t xdr_opaque(xdrs, cp, cnt)
XDR *xdrs;
char *cp;
u_int cnt;
```

A filter primitive that translates between fixed size opaque data and its external representation. The parameter *cp* is the address of the opaque object, and *cnt* is its size in bytes. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_pointer(xdrs, objpp, objsize, objproc)
XDR *xdrs;
char **objpp;
u_int objsize;
xdrproc_t objproc;

Like **xdr_reference()** except that it serializes NULL pointers, whereas **xdr_reference()** does not. Thus, **xdr_pointer()** can represent recursive data structures, such as binary trees or linked lists. The parameter *objpp* is the address of the pointer; *objsize* is the *sizeof* the structure that **objpp* points to; and *objproc* is an XDR procedure that filters the structure between its C form and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_reference(xdrs, pp, size, proc)
XDR *xdrs;
char **pp;
u_int size;
xdrproc_t proc;

A primitive that provides pointer chasing within structures. The parameter *pp* is the address of the pointer; *size* is the *sizeof* the structure that **pp* points to; and *proc* is an XDR procedure that filters the structure between its C form and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

Warning: This routine does not understand NULL pointers. Use **xdr_pointer()** instead.

bool_t xdr_string(xdrs, strp, maxsize)
XDR *xdrs;
char **strp;
u_int maxsize;

A filter primitive that translates between C strings and their corresponding external representations. The routine fails if the string being translated is longer than *maxsize*. *strp* is the address of the pointer to the string. While decoding if **strp* is NULL, then the necessary storage is allocated to hold this null-terminated string and **strp* is set to point to this. This storage can be freed by using **xdr_free()** (see **xdr_simple**(3N)). This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_union(xdrs, dscmp, unp, choices, defaultarm)
XDR *xdrs;
int *dscmp;
char *unp;
struct xdr_discrim *choices;
bool_t (*defaultarm) (); /* may be NULL */

A filter primitive that translates between a discriminated C **union** and its corresponding external representation. It first translates the discriminant of the union located at *dscmp*. This discriminant is always an **enum_t**. Next the union located at *unp* is translated. The parameter *choices* is a pointer to an array of **xdr_discrim** structures. Each structure contains an ordered pair of [*value*,*proc*]. If the union's discriminant is equal to any of the *value*s, then the associated *proc* is called to translate the union. The end of the **xdr_discrim** structure array is denoted by a NULL pointer. If the discriminant is not found in the *choices* array, then the *defaultarm* procedure is called (if it is not NULL). This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_vector(xdrs, arrp, size, elsize, elproc)
XDR *xdrs;
char *arrp;
u_int size, elsize;
xdrproc_t elproc;

A filter primitive that translates between fixed-length arrays and their corresponding external representations. The parameter *arrp* is the address of the array, while *size* is the element count of the array. The parameter *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_wrapstring(xdrs, strp)
XDR *xdrs;
char **strp;

A primitive that calls **xdr_string(xdrs, strp, MAXUNSIGNED);** where **MAXUNSIGNED** is the maximum value of an unsigned integer. **xdr_wrapstring( )** is handy because the RPC package passes a maximum of two XDR routines as parameters, and **xdr_string( )**, one of the most frequently used primitives, requires three. *strp* is the address of the pointer to the string. While decoding if **strp* is NULL, then the necessary storage is allocated to hold the null-terminated string and **strp* is set to point to this. This storage can be freed by using **xdr_free( )** (see **xdr_simple**(3N)). This routine returns TRUE if it succeeds, FALSE otherwise.

SEE ALSO

xdr(3N), **xdr_admin**(3N), **xdr_create**(3N), **xdr_simple**(3N)

NAME

xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation

DESCRIPTION

XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.

These routines deal with the creation of XDR streams. XDR streams have to be created before any data can be translated into XDR format.

Routines

The **XDR**, **CLIENT**, and **SVCXPRT** data structures are defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/xdr.h>

void xdr_destroy(xdrs)
XDR *xdrs;

Invoke the destroy routine associated with the XDR stream, *xdrs*. Destruction usually involves freeing private data structures associated with the stream. Using *xdrs* after invoking **xdr_destroy()** is undefined.

void xdrmem_create(xdrs, addr, size, op)
XDR *xdrs;
char *addr;
u_int size;
enum xdr_op op;

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to, or read from, a chunk of memory at location *addr* whose length is no more than *size* bytes long. *size* should be a multiple of 4. The *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

void xdrrec_create(xdrs, sendsz, recvsz, handle, readit, writeit)
XDR *xdrs;
u_int sendsz, recvsz;
char *handle;
int (*readit) (), (*writeit) ();

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to a buffer of size *sendsz*; a value of zero indicates the system should use a suitable default. The stream's data is read from a buffer of size *recvsz*; it too can be set to a suitable default by passing a zero value. When a stream's output buffer is full, *writeit* is called. Similarly, when a stream's input buffer is empty, *readit* is called. The behavior of these two routines is similar to **read**(2V) and **write**(2V), except that *handle* is passed to the former routines as the first parameter. Note: The XDR stream's *op* field must be set by the caller. *sendsz* and *recvsz* should be multiples of 4.

Warning: This XDR stream implements an intermediate record stream. Therefore there are additional bytes in the stream to provide record boundary information.

void xdrstdio_create(xdrs, filep, op)
XDR *xdrs;
FILE *filep;
enum xdr_op op;

This routine initializes the XDR stream object pointed to by *xdrs*. The XDR stream data is written to, or read from, the Standard **I/O** stream *filep*. The parameter *op* determines the direction of the XDR stream (either **XDR_ENCODE, XDR_DECODE,** or **XDR_FREE**).

Warning: The destroy routine associated with such XDR streams calls **fflush()** on the *file* stream, but never **fclose**(3V).

SEE ALSO

read(2V), **write**(2V), **fclose**(3V), **xdr**(3N), **xdr_admin**(3N), **xdr_complex**(3N), **xdr_simple**(3N)

NAME

xdr_bool, xdr_char, xdr_double, xdr_enum, xdr_float, xdr_free, xdr_int, xdr_long, xdr_short, xdr_u_char, xdr_u_int, xdr_u_long, xdr_u_short, xdr_void – library routines for translating simple data types

DESCRIPTION

XDR library routines allow C programmers to describe simple data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.

These routines require the creation of XDR streams (see **xdr_create**(3N)).

Routines

The **XDR** data structure is defined in the RPC/XDR Library Definitions of the *Network Programming*.

#include <rpc/xdr.h>

bool_t xdr_bool(xdrs, bp)
XDR *xdrs;
bool_t *bp;

A filter primitive that translates between a boolean (C integer) and its external representation. When encoding data, this filter produces values of either one or zero. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_char(xdrs, cp)
XDR *xdrs;
char *cp;

A filter primitive that translates between a C character and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

Note: Encoded characters are not packed, and occupy 4 bytes each. For arrays of characters, it is worthwhile to consider **xdr_bytes()**, **xdr_opaque()** or **xdr_string()** , see **xdr_complex**(3N).

bool_t xdr_double(xdrs, dp)
XDR *xdrs;
double *dp;

A filter primitive that translates between a C **double** precision number and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_enum(xdrs, ep)
XDR *xdrs;
enum_t *ep;

A filter primitive that translates between a C **enum** (actually integer) and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_float(xdrs, fp)
XDR *xdrs;
float *fp;

A filter primitive that translates between a C **float** and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

void xdr_free(proc, objp)
xdrproc_t proc;
char *objp;

Generic freeing routine. The first argument is the XDR routine for the object being freed. The second argument is a pointer to the object itself. Note: The pointer passed to this routine is *not* freed, but what it points to *is* freed, recursively such that objects pointed to are also freed for example, linked lists.

bool_t xdr_int(xdrs, ip)
XDR *xdrs;
int *ip;

A filter primitive that translates between a C **integer** and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_long(xdrs, lp)
XDR *xdrs;
long *lp;

A filter primitive that translates between a C **long** integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_short(xdrs, sp)
XDR *xdrs;
short *sp;

A filter primitive that translates between a C **short** integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_char(xdrs, ucp)
XDR *xdrs;
unsigned char *ucp;

A filter primitive that translates between an **unsigned** C character and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_int(xdrs, up)
XDR *xdrs;
unsigned *up;

A filter primitive that translates between a C **unsigned** integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_long(xdrs, ulp)
XDR *xdrs;
unsigned long *ulp;

A filter primitive that translates between a C **unsigned long** integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_short(xdrs, usp)
XDR *xdrs;
unsigned short *usp;

A filter primitive that translates between a C **unsigned short** integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_void()

This routine always returns TRUE. It may be passed to RPC routines that require a function parameter, where nothing is to be done.

SEE ALSO

xdr(3N), **xdr_admin**(3N), **xdr_complex**(3N), **xdr_create**(3N)

NAME

ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err – NIS client interface

SYNOPSIS AND DESCRIPTION

This package of functions provides an interface to the Network Information Service (NIS). The package can be loaded from the standard library, **/usr/lib/libc.a**. Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS name service, including the definitions of *map* and *domain*, and a description of the various servers, databases, and commands that comprise the NIS services.

All input parameters names begin with *in*. Output parameters begin with *out*. Output parameters of type **char **** should be addresses of uninitialized character pointers. Memory is allocated by the NIS client package using **malloc**(3V), and may be freed if the user code has no continuing need for it. For each *outkey* and *outval*, two extra bytes of memory are allocated at the end that contain NEWLINE and the null character, respectively, but these two bytes are not reflected in *outkeylen* or *outvallen*. *indomain* and *inmap* strings must not be empty and must be null-terminated. String parameters which are accompanied by a count parameter may not be NULL, but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions in this package of type *int* return 0 if they succeed, and a failure code (**YPERR_***xxxx*) otherwise. Failure codes are described under DIAGNOSTICS below.

yp_bind (indomain);
char *indomain;

To use the NIS services, the client process must be "bound" to a NIS server that serves the appropriate domain using **yp_bind()**. Binding need not be done explicitly by user code; this is done automatically whenever a NIS lookup function is called. **yp_bind()** can be called directly for processes that make use of a backup strategy (for example, a local file) in cases when NIS services are not available.

void
yp_unbind (indomain)
char *indomain;

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. **yp_unbind()** is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to **yp_unbind()** make the domain *unbound*, and free all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the ypclnt layer will retry forever or until the operation succeeds, provided that **ypbind** is running, and either

a) the client process cannot bind a server for the proper domain, or

b) RPC requests to the server fail.

If an error is not RPC-related, or if **ypbind** is not running, or if a bound **ypserv** process returns any answer (success or failure), the ypclnt layer will return control to the user code, either with an error code, or a success code and any results.

yp_get_default_domain (outdomain);
char **outdomain;

The NIS lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling **yp_get_default_domain( )**, and use the returned *outdomain* as the *indomain* parameter to successive NIS calls.

yp_match(indomain, inmap, inkey, inkeylen, outval, outvallen)
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outval;
int *outvallen;

yp_match( ) returns the value associated with a passed key. This key must be exact; no pattern matching is available.

yp_first(indomain, inmap, outkey, outkeylen, outval, outvallen)
char *indomain;
char *inmap;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_first( ) returns the first key-value pair from the named map in the named domain.

yp_next(indomain, inmap, inkey, inkeylen, outkey, outkeylen, outval, outvallen);
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_next( ) returns the next key-value pair in a named map. The *inkey* parameter should be the *outkey* returned from an initial call to **yp_first( )** (to get the second key-value pair) or the one returned from the nth call to **yp_next( )** (to get the nth + second key-value pair).

The concept of first (and, for that matter, of next) is particular to the structure of the NIS map being processing; there is no relation in retrieval order to either the lexical order within any original (non-NIS) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the **yp_first( )** function is called on a particular map, and then the **yp_next( )** function is repeatedly called on the same map at the same server until the call fails with a reason of **YPERR_NOMORE**, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all(indomain, inmap, incallback);
char *indomain;
char *inmap;
struct ypall_callback *incallback;

yp_all() provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. You can use **yp_all()** just like any other NIS procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to **yp_all()** only when the transaction is completed (successfully or unsuccessfully), or your foreach function decides that it does not want to see any more key-value pairs.

The third parameter to **yp_all()** is

```
struct ypall_callback *incallback {
int (*foreach)();
char *data;
};
```

The function **foreach** is called

```
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
int inkeylen;
char *inval;
int invallen;
char *indata;
```

The *instatus* parameter will hold one of the return status values defined in **<rpcsvc/yp_prot.h>** — either **YP_TRUE** or an error code. See **ypprot_err()**, below, for a function which converts a NIS protocol error code to a ypclnt layer error code.

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the **yp_all()** function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the **foreach** function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the **foreach** function look exactly as they do in the server's map — if they were not NEWLINE-terminated or null-terminated in the map, they will not be here either.

The *indata* parameter is the contents of the **incallback->data** element passed to **yp_all()**. The **data** element of the callback structure may be used to share state information between the **foreach** function and the mainline code. Its use is optional, and no part of the NIS client package inspects its contents — cast it to something useful, or ignore it as you see fit.

The **foreach** function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If **foreach** returns a non-zero value, it is not called again; the functional value of **yp_all()** is then 0.

yp_order(indomain, inmap, outorder);
char *indomain;
char *inmap;
int *outorder;

yp_order( ) returns the order number for a map.

yp_master(indomain, inmap, outname);
char *indomain;
char *inmap;
char **outname;

yp_master( ) returns the machine name of the master NIS server for a map.

char *yperr_string(incode)
int incode;

yperr_string( ) returns a pointer to an error message string that is null-terminated but contains no period or NEWLINE.

ypprot_err (incode)
unsigned int incode;

ypprot_err( ) takes a NIS protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to **yperr_string( )**.

FILES

<rpcsvc/ypclnt.h>
<rpcsvc/yp_prot.h>
/usr/lib/libc.a

SEE ALSO

malloc(3V), **ypupdate**(3N), **ypfiles**(5), **ypserv**(8)

DIAGNOSTICS

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

```
#define YPERR_BADARGS
        1    /* args to function are bad */

#define YPERR_RPC
        2    /* RPC failure - domain has been unbound */

#define YPERR_DOMAIN
        3    /* can't bind to server on this domain */

#define YPERR_MAP
        4    /* no such map in server's domain */

#define YPERR_KEY
        5    /* no such key in map */

#define YPERR_YPERR
        6    /* internal yp server or client error */

#define YPERR_RESRC
        7    /* resource allocation failure */

#define YPERR_NOMORE
        8    /* no more records in map database */

#define YPERR_PMAP
        9    /* can't communicate with portmapper */

#define YPERR_YPBIND
```

```
        10      /* can't communicate with ypbind */
#define YPERR_YPSERV
        11      /* can't communicate with ypserv */
#define YPERR_NODOM
        12      /* local domain name not set */
#define YPERR_BADDBfR
        13      /* yp database is bad */
#define YPERR_VERSfR
        14      /* yp version mismatch */
#define YPERR_ACCESS
        15      /* access violation */
#define YPERR_BUSY
        16      /* database busy */
```

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yp_update – changes NIS information

SYNOPSIS

#include <rpcsvc/ypclnt.h>

yp_update(domain, map, ypop, key, keylen, data, datalen)
char *domain;
char *map;
unsigned ypop
char *key;
int keylen;
char *data;
int datalen;

DESCRIPTION

yp_update( ) is used to make changes to the Network Information Service (NIS) database. The syntax is the same as that of **yp_match**( ) (see **ypclnt**(3N)) except for the extra parameter *ypop* which may take on one of four values. If it is **YPOP_CHANGE** then the data associated with the key will be changed to the new value. If the key is not found in the database, then **yp_update()** returns **YPERR_KEY**. If *ypop* has the value **YPOP_INSERT** then the key-value pair will be inserted into the database. The error **YPERR_KEY** is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass *ypop* as **YPOP_STORE** and no error will be returned if the key already or does not exist. To delete an entry, the value of *ypop* should be **YPOP_DELETE**.

This routine depends upon secure RPC, and will not work unless the network is running secure RPC.

SEE ALSO

ypclnt(3N)

System and Network Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

intro – introduction to the lightweight process library (LWP)

DESCRIPTION

The lightweight process library (LWP) provides a mechanism to support multiple threads of control that share a single address space. Under SunOS, the address space is derived from a single *forked* ("heavyweight") process. Each thread has its own stack segment (specified when the thread is created) so that it can access local variables and make procedure calls independently of other threads. The collection of threads sharing an address space is called a *pod*. Under SunOS, threads share all of the resources of the heavyweight process that contains the pod, including descriptors and signal handlers.

The LWP provides a means for creating and destroying threads, message exchange between threads, manipulating condition variables and monitors, handling synchronous exceptions, mapping asynchronous events into messages, mapping synchronous events into exceptions, arranging for special per-thread context, multiplexing the clock for timeouts, and scheduling threads both preemptively and non–preemptively.

The LWP system exists as a library of routines (**/usr/lib/liblwp.a**) linked in (**–llwp**) with a client program which should **#include** the file **<lwp/lwp.h>**. **main** is transparently converted into a lightweight process as soon as it attempts to use any LWP primitives.

When an object created by a LWP primitive is destroyed, every attempt is made to clean up after it. For example, if a thread dies, all threads blocked on sends to or receives from that thread are unblocked, and all monitor locks held by the dead thread are released.

Because there is no kernel support for threads at present, system calls effectively block the entire pod. By linking in the non-blocking I/O library (**–lnbio**) ahead of the LWP library, you can alleviate this problem for those system calls that can issue a signal when a system call would be profitable to try. This library (which redefines some system calls) uses asynchronous I/O and events (for example, **SIGCHLD** and **SIGIO**) to make blocking less painful. The system calls remapped by the nbio library are: **open**(2V), **socket**(2), **pipe**(2V), **close**(2V), **read**(2V), **write**(2V), **send**(2), **recv**(2), **accept**(2), **connect**(2), **select (2)** and **wait**(2V).

RETURN VALUES

LWP primitives return non-negative integers on success. On errors, they return –1. See **lwp_perror**(3L) for details on error handling.

FILES

/usr/lib/liblwp.a
/usr/lib/libnbio.a

SEE ALSO

accept(2), **close**(2V), **connect**(2), **open**(2V), **pipe**(2V), **read**(2V), **recv**(2), **select**(2), **send**(2), **socket**(2), **wait**(2V) **write**(2V)
Lightweight Processes in the *System Services Overview*

INDEX

The following are the primitives currently supported, grouped roughly by function.

Thread Creation

lwp_self(tid)
lwp_getstate(tid, statvec)
lwp_setregs(tid, machstate)
lwp_getregs(tid, machstate)
lwp_ping(tid)
lwp_create(tid, pc, prio, flags, stack, nargs, arg1, . . . , argn)
lwp_destroy(tid)
lwp_enumerate(vec, maxsize)
pod_setexit(status)
pod_getexit( )
pod_exit(status)
SAMETHREAD(t1, t2)

Thread Scheduling

pod_setmaxpri(maxprio)
pod_getmaxpri()
pod_getmaxsize()
lwp_resched(prio)
lwp_setpri(tid, prio)
lwp_sleep(timeout)
lwp_suspend(tid)
lwp_resume(tid)
lwp_yield(tid)
lwp_join(tid)

Error Handling

lwp_geterr()
lwp_perror(s)
lwp_errstr()

Messages

msg_send(tid, argbuf, argsize, resbuf, ressize)
msg_recv(tid, argbuf, argsize, resbuf, ressize, timeout)
MSG_RECVALL(tid, argbuf, argsize, resbuf, ressize, timeout)
msg_reply(tid)
msg_enumsend(vec, maxsize)
msg_enumrecv(vec, maxsize)

Event Mapping (Agents)

agt_create(agt, event, memory)
agt_enumerate(vec, maxsize)
agt_trap(event)

Thread Synchronization: Monitors

mon_create(mid)
mon_destroy(mid)
mon_enter(mid)
mon_exit(mid)
mon_enumerate(vec, maxsize)
mon_waiters (mid, owner, vec, maxsize)
mon_cond_enter(mid)
mon_break(mid)
MONITOR(mid)
SAMEMON(m1, m2)

Thread Synchronization: Condition Variables

cv_create(cv, mid)
cv_destroy(cv)
cv_wait(cv)
cv_notify(cv)
cv_send(cv, tid)
cv_broadcast(cv)
cv_enumerate(vec, maxsize)
cv_waiters(cv, vec, maxsize)
SAMECV(c1, c2)

Exception Handling

exc_handle(pattern, func, arg)
exc_unhandle()
(*exc_bound(pattern, arg))()
exc_notify(pattern)
exc_raise(pattern)

exc_on_exit(func, arg)
exc_uniqpatt( )

Special Context Handling

lwp_ctxinit(tid, cookie)
lwp_ctxremove(tid, cookie)
lwp_ctxset(save, restore, ctxsize, optimise)
lwp_ctxmemget(mem, tid, ctx)
lwp_ctxmemset(mem, tid, ctx)
lwp_fpset(tid)
lwp_libcset(tid)

Stack Management

CHECK(location, result)
lwp_setstkcache(minsize, numstks)
lwp_newstk( )
lwp_datastk(data, size, addr)
lwp_stkcswset(tid, limit)
lwp_checkstkset(tid, limit)
STKTOP(s)

BUGS

There is no language support available from C.

There is no kernel support yet. Thus system calls in different threads cannot execute in parallel.

Killing a process that uses the non-blocking I/O library may leave objects (such as its standard input) in a non-blocking state. This could cause confusion to the shell.

LIST OF LWP LIBRARY FUNCTIONS

Name	Appears on Page	Description
agt_create	**agt_create**(3L)	map LWP events into messages
agt_enumerate	**agt_create**(3L)	map LWP events into messages
agt_trap	**agt_create**(3L)	map LWP events into messages
CHECK	**lwp_newstk**(3L)	LWP stack management
cv_broadcast	**cv_create**(3L)	manage LWP condition variables
cv_create	**cv_create**(3L)	manage LWP condition variables
cv_destroy	**cv_create**(3L)	manage LWP condition variables
cv_enumerate	**cv_create**(3L)	manage LWP condition variables
cv_notify	**cv_create**(3L)	manage LWP condition variables
cv_send	**cv_create**(3L)	manage LWP condition variables
cv_wait	**cv_create**(3L)	manage LWP condition variables
cv_waiters	**cv_create**(3L)	manage LWP condition variables
exc_bound	**exc_handle**(3L)	LWP exception handling
exc_handle	**exc_handle**(3L)	LWP exception handling
exc_notify	**exc_handle**(3L)	LWP exception handling
exc_on_exit	**exc_handle**(3L)	LWP exception handling
exc_raise	**exc_handle**(3L)	LWP exception handling
exc_unhandle	**exc_handle**(3L)	LWP exception handling
exc_uniqpatt	**exc_handle**(3L)	LWP exception handling
lwp_checkstkset	**lwp_newstk**(3L)	LWP stack management
lwp_create	**lwp_create**(3L)	LWP thread creation and destruction primitives
lwp_ctxinit	**lwp_ctxinit**(3L)	special LWP context operations
lwp_ctxmemget	**lwp_ctxinit**(3L)	special LWP context operations
lwp_ctxmemset	**lwp_ctxinit**(3L)	special LWP context operations
lwp_ctxremove	**lwp_ctxinit**(3L)	special LWP context operations
lwp_ctxset	**lwp_ctxinit**(3L)	special LWP context operations
lwp_datastk	**lwp_newstk**(3L)	LWP stack management

lwp_destroy	**lwp_create**(3L)	LWP thread creation and destruction primitives
lwp_enumerate	**lwp_status**(3L)	LWP status information
lwp_errstr	**lwp_perror**(3L)	LWP error handling
lwp_fpset	**lwp_ctxinit**(3L)	special LWP context operations
lwp_geterr	**lwp_perror**(3L)	LWP error handling
lwp_getregs	**lwp_status**(3L)	LWP status information
lwp_getstate	**lwp_status**(3L)	LWP status information
lwp_join	**lwp_yield**(3L)	control LWP scheduling
lwp_libcset	**lwp_ctxinit**(3L)	special LWP context operations
lwp_newstk	**lwp_newstk**(3L)	LWP stack management
lwp_perror	**lwp_perror**(3L)	LWP error handling
lwp_ping	**lwp_status**(3L)	LWP status information
lwp_resched	**lwp_yield**(3L)	control LWP scheduling
lwp_resume	**lwp_yield**(3L)	control LWP scheduling
lwp_self	**lwp_status**(3L)	LWP status information
lwp_setpri	**lwp_yield**(3L)	control LWP scheduling
lwp_setregs	**lwp_status**(3L)	LWP status information
lwp_setstkcache	**lwp_newstk**(3L)	LWP stack management
lwp_sleep	**lwp_yield**(3L)	control LWP scheduling
lwp_stkcswset	**lwp_newstk**(3L)	LWP stack management
lwp_suspend	**lwp_yield**(3L)	control LWP scheduling
lwp_yield	**lwp_yield**(3L)	control LWP scheduling
MINSTACKSZ	**lwp_newstk**(3L)	LWP stack management
mon_break	**mon_create**(3L)	LWP routines to manage critical sections
mon_cond_enter	**mon_create**(3L)	LWP routines to manage critical sections
mon_create	**mon_create**(3L)	LWP routines to manage critical sections
mon_destroy	**mon_create**(3L)	LWP routines to manage critical sections
mon_enter	**mon_create**(3L)	LWP routines to manage critical sections
mon_enumerate	**mon_create**(3L)	LWP routines to manage critical sections
mon_exit	**mon_create**(3L)	LWP routines to manage critical sections
mon_waiters	**mon_create**(3L)	LWP routines to manage critical sections
MONITOR	**mon_create**(3L)	LWP routines to manage critical sections
msg_enumrecv	**msg_send**(3L)	LWP send and receive messages
msg_enumsend	**msg_send**(3L)	LWP send and receive messages
msg_recv	**msg_send**(3L)	LWP send and receive messages
MSG_RECVALL	**msg_send**(3L)	LWP send and receive messages
msg_reply	**msg_send**(3L)	LWP send and receive messages
msg_send	**msg_send**(3L)	LWP send and receive messages
pod_exit	**lwp_create**(3L)	LWP thread creation and destruction primitives
pod_getexit	**lwp_create**(3L)	LWP thread creation and destruction primitives
pod_getmaxpri	**pod_getmaxpri**(3L)	control LWP scheduling priority
pod_getmaxsize	**pod_getmaxpri**(3L)	control LWP scheduling priority
pod_setexit	**lwp_create**(3L)	LWP thread creation and destruction primitives
pod_setmaxpri	**pod_getmaxpri**(3L)	control LWP scheduling priority
SAMECV	**cv_create**(3L)	manage LWP condition variables
SAMEMON	**mon_create**(3L)	LWP routines to manage critical sections
SAMETHREAD	**lwp_create**(3L)	LWP thread creation and destruction primitives
STKTOP	**lwp_newstk**(3L)	LWP stack management

NAME

agt_create, agt_enumerate, agt_trap – map LWP events into messages

SYNOPSIS

#include <lwp/lwp.h>

thread_t agt_create(agt, event, memory)
thread_t *agt;
int event;
caddr_t memory;

int agt_enumerate(vec, maxsize)
thread_t vec[ ];
int maxsize;

int agt_trap(event)
int event;

DESCRIPTION

Agents are entities that act like threads sending messages when an asynchronous event occurs. **agt_create()** creates an object called an *agent* which maps the asynchronous event *event* into messages that can be received with **msg_recv()** (see **msg_send**(3L)). *agt* stores the handle on this object. *event* is a UNIX signal number.

agt_trap() causes the event, *event*, to generate an exception (see **exc_handle**(3L)). Once initialized using **agt_create()** or **agt_trap()**, an event can not be remapped to a different style of handling. If traps are enabled, an event will cause the termination of the *thread* running at the time of the trap if the trap exception is not handled. If an exception handler is in place, an exception will be raised. If an agent exists for the event, the event is mapped into a message for the agent. If neither agent nor trap mapping is enabled, the default signal action (SIG_DFL) is applied to the *pod*. Use of standard UNIX signal handling facilities will defeat the event mapping mechanism.

The message sent by the agent (in the argument buffer) will look like any other message with the sender being the agent. The receive buffer is NULL. A message is always sent by an agent to the thread which created the agent.

All messages sent by an agent contain an **eventinfo_t**. This structure indicates the thread running at the time the interrupt happened, and the particular event that occurred. Some agent messages contain more information if the particular event warrants it. In this case, a struct containing an **eventinfo_t** as its first element is passed as the argument buffer. Definitions of these structures are contained in **<lwp/lwp.h>**.

An agent appears to the owning thread just like another thread. It must therefore have some memory for holding its message, as the sender and receiver must belong to the same address space. *memory* is the space an agent will use to store its message. Typically, this is on the stack of the thread that created the agent. It must be of the correct size for the kind of event being created (most events need something to store an **eventinfo_t**. **SIGCHLD** events need room for a **sigchldev_t**.)

You should reply to an agent (using **msg_reply()** (see **msg_send**(3L)) as you would reply to a thread. Although agents do not ordinarily lose events, the next agent message will not be delivered until a reply is sent to the agent. Thus, an agent appears to the client as an ordinary thread sending messages. An agent will only lose events if the total number of unreplied-to events in a pod exceeds AGENTMEMORY.

lwp_destroy() is used to destroy an agent. All agents created by a thread automatically disappear when that thread dies. **agt_enumerate()** fills in a list with the ID's of all existing agents and returns the total number of agents. This primitive uses *maxsize* to avoid exceeding the capacity of the list. If the number of agents is greater than *maxsize*, only *maxsize* agents ID's are filled in *vec*. If *maxsize* is zero, **agt_enumerate()** returns the total number of agents.

The special event **LASTRITES** is caused by the termination of a thread. An agent for **LASTRITES** will be informed about every thread that terminates, regardless of cause. The **eventinfo_code** element of this agent will contain the stack argument that the dead thread was created with. Note: by allocating adjacent space above the thread stack, this argument can be used to point to private information about a thread. The **eventinfo_victimid** element will contain the id of the dead thread.

RETURN VALUES

agt_create( ) and **agt_trap**( ) return:

0	on success.
−1	on failure.

agt_enumerate( ) returns the total number of agents.

ERRORS

agt_trap( ) will fail if one or more of the following are true:

LE_INUSE	Agent in use for this event.
LE_INVALIDARG	Event specified does not exist.

agt_create( ) will fail if one or more of the following are true:

LE_INUSE	Trap mapping in use for this event.
LE_INVALIDARG	Attempt to create agent for non-existent event.

SEE ALSO

exc_handle(3L), **msg_send**(3L)

BUGS

Signal handlers always take the **SIG_DFL** action when no agent manages the event.

If a descriptor used by a parent of the pod (such as its standard input) is marked non-blocking by a thread, it should be reset when the pod terminates to prevent the parent from receiving EWOULDBLOCK errors on the descriptor. There is no way to prevent this from happening if a pod is terminated with extreme prejudice (for instance, using **SIGKILL**).

If an agent reports that a descriptor has I/O available, there may be more than one occurrence of I/O available from that descriptor. Thus, being informed that **SIGIO** has occurred on socket *s* may mean that there are several messages waiting to be received from *s*. Clients should be careful to clean out all I/O from a descriptor before going back to sleep.

All system calls should be protected with loops testing for EINTR (and monitors if multiple threads can try to use system calls concurrently). An **lwp_sleep**( ) could result in a hidden clock interrupt for example.

WARNINGS

agt_trap( ) should not be used for asynchronous events. If an unsuspecting thread which has no exception handler is running at the time of a trapped event, it will be terminated.

Clients should not normally handle signals themselves since the agent mechanism assumes it is the only entity handling signals.

NAME

cv_create, cv_destroy, cv_wait, cv_notify, cv_broadcast, cv_send, cv_enumerate, cv_waiters, SAMECV – manage LWP condition variables

SYNOPSIS

```
#include <lwp/lwp.h>

cv_t cv_create(cv, mid)
cv_t *cv;
mon_t mid;

int cv_destroy(cv)
cv_t cv;

int cv_wait(cv)
cv_t cv;

int cv_notify(cv)
cv_t cv;

int cv_send(cv, tid)
cv_t cv;
lwp_t tid

int cv_broadcast(cv)
cv_t cv;

int cv_enumerate(vec, maxsize)
cv_t vec[ ];        /* will contain list of all conditions */
int maxsize;        /* maximum size of vec */

int cv_waiters(cv, vec, maxsize)
cv_t cv;            /* condition variable being interrogated */
thread_t vec[ ];    /* which threads are blocked on cv */
int maxsize;        /* maximum size of vec */

SAMECV(c1, c2)
```

DESCRIPTION

Condition variables are useful for synchronization within monitors. By waiting on a condition variable, the currently-held monitor (a condition variable must *always* be used within a monitor) is released atomically and the invoking thread is suspended. When monitors are nested, monitor locks other than the current one are retained by the thread. At some later point, a different thread may awaken the waiting thread by issuing a notification on the condition variable. When the notification occurs, the waiting thread will queue to reacquire the monitor it gave up. It is possible to have different condition variables operating within the same monitor to allow selectivity in waking up threads.

cv_create( ) creates a new condition variable (returned in *cv*) which is bound to the monitor specified by *mid*. It is illegal to access (using **cv_wait( )**, **cv_notify( )**, **cv_send( )** or **cv_broadcast( )**) a condition variable from a monitor other than the one it is bound to. **cv_destroy( )** removes a condition variable.

cv_wait( ) blocks the current thread and releases the monitor lock associated with the condition (which must also be the monitor lock most recently acquired by the thread). Other monitor locks held by the thread are not affected. The blocked thread is enqueued by its scheduling priority on the condition.

cv_notify( ) awakens at most one thread blocked on the condition variable and causes the awakened thread to queue for access to the monitor released at the time it waited on the condition. It can be dangerous to use **cv_notify( )** if there is a possibility that the thread being awakened is one of several threads that are waiting on a condition variable and the awakened thread may not be the one intended. In this case, use of **cv_broadcast( )** is recommended.

cv_broadcast() is the same as **cv_notify()** except that *all* threads blocked on the condition variable are awakened. **cv_notify()** and **cv_broadcast()** do nothing if no thread is waiting on the condition. For both **cv_notify()** and **cv_broadcast()**, the currently held monitor must agree with the one bound to the condition by **cv_create()**.

cv_send() is like **cv_notify()** except that the particular thread **tid** is awakened. If this thread is not currently blocked on the condition, **cv_send()** reports an error.

cv_enumerate() lists the ID of all of the condition variables. The value returned is the total number of condition variables. The vector supplied is filled in with the ID's of condition variables. **cv_waiters()** lists the ID's of the threads blocked on the condition variable *cv* and returns the number of threads blocked on *cv*. For both **cv_enumerate()** and **cv_waiters()**, *maxsize* is used to avoid exceeding the capacity of the list *vec*. If the number of entries to be filled is greater than *maxsize*, only *maxsize* entries are filled in *vec*. It is legal in both of these primitives to specify a *maxsize* of 0.

SAMECV is a convenient predicate used to compare two condition variables for equality.

RETURN VALUES

cv_create(), **cv_destroy()**, **cv_send()**, **cv_wait()**, **cv_notify()** and **cv_broadcast()** return:

0 on success.

−1 on failure and set **errno** to indicate the error.

cv_enumerate() returns the total number of condition variables.

cv_waiters() returns the number of threads blocked on a condition variable.

ERRORS

cv_destroy() will fail if one or more of the following is true:

LE_INUSE Attempt to destroy condition variable being waited on by a thread.

LE_NONEXIST Attempt to destroy non-existent condition variable.

cv_wait() will fail if one or more of the following is true:

LE_NONEXIST Attempt to wait on non-existent condition variable.

LE_NOTOWNED Attempt to wait on a condition without possessing the correct monitor lock.

cv_notify() will fail if one or more of the following is true:

LE_NONEXIST Attempt to notify non-existent condition variable.

LE_NOTOWNED Attempt to notify condition variable without possessing the correct monitor.

cv_send() will fail if one or more of the following is true:

LE_NONEXIST Attempt to awaken non-existent condition variable.

LE_NOTOWNED Attempt to awaken condition variable without possessing the correct monitor lock.

LE_NOWAIT The specified thread is not currently blocked on the condition.

cv_broadcast() will fail if one or more of the following is true:

LE_NONEXIST Attempt to broadcast non-existent condition variable.

LE_NOTOWNED Attempt to broadcast condition without possessing the correct monitor lock.

SEE ALSO

mon_create(3L)

NAME

exc_handle, exc_unhandle, exc_bound, exc_notify, exc_raise, exc_on_exit, exc_uniqpatt – LWP exception handling

SYNOPSIS

```
#include <lwp/lwp.h>

int exc_handle(pattern, func, arg)
int pattern;
caddr_t (*func)( );
caddr_t arg;

int exc_raise(pattern)
int pattern;

int exc_unhandle( )

caddr_t (*exc_bound(pattern, arg))( )
int pattern;
caddr_t *arg;

int exc_notify(pattern)
int pattern;

int exc_on_exit(func, arg)
void (*func)( );
caddr_t arg;

int exc_uniqpatt( )
```

DESCRIPTION

These primitives can be used to manage exceptional conditions in a thread. Basically, raising an exception is a more general form of non-local goto or *longjmp*, but the invocation is pattern-based. It is also possible to *notify* an exception handler whereby a function supplied by the exception handler is invoked and control is returned to the raiser of the exception. Finally, one can establish a handler which is always invoked upon procedure exit, regardless of whether the procedure exits using a *return* or an exception raised to a handler established prior to the invocation of the exiting procedure.

exc_handle( ) is used to establish an exception handler. **exc_handle( )** returns 0 to indicate that a handler has been established. A return of −1 indicates an error in trying to establish the exception handler. If it returns something else, an exception has occurred and any procedure calls deeper than the one containing the handler have disappeared. All exception handlers established by a procedure are automatically discarded when the procedure terminates.

exc_handle( ) binds a *pattern* to the handler, where a pattern is an integer, and two patterns *match* if their values are equal. When an exception is raised with **exc_raise( )**, the most recent handler that has established a matching pattern will catch the exception. A special pattern (CATCHALL) is provided which matches any **exc_raise( )** pattern. This is useful for handlers which know that there is no chance the resources allocated in a routine can be reclaimed by previous routines in the call chain.

The other two arguments to **exc_handle( )** are a function and an argument to that function. **exc_bound( )** retrieves these arguments from an **exc_handle( )** call made by the specified thread. By using **exc_bound( )** to retrieve and call a function bound by the exception handler, a procedure can raise a *notification exception* which allows control to return to the raiser of the exception after the exception is handled.

exc_raise( ) allows the caller to transfer control (do a non-local goto) to the matching **exc_handle( )**. This matching exception handler is destroyed after the control transfer. At this time, it behaves as if **exc_handle( )** returns with the *pattern* from **exc_raise( )** as the return value. Note: *func* of **exc_handle( )** is not called using **exc_raise( )** — it is only there for notification exceptions. Because the exception handler returns the pattern that invoked it, it is possible for a handler that matches the CATCHALL pattern to *reraise* the exact exception it caught by using **exc_raise( )** on the caught pattern. It is illegal to handle or raise the pattern 0 or the pattern −1. Handlers are searched for pattern matches in the reverse execution order that they are set (i.e., the most recently established handler is searched first).

exc_unhandle( ) destroys the most recently established exception handler set by the current thread. It is an error to destroy an exit-handler set up by **exc_on_exit( )**. When a procedure exits, all handlers and exit handlers set in the procedure are automatically deallocated.

exc_notify( ) is a convenient way to use **exc_bound**. The function which is bound to *pattern* is retrieved. If the function is not NULL, the function is called with the associated argument and the result is returned. If the function is NULL, **exc_raise(*pattern*)** is returned.

exc_on_exit( ) specifies an exit procedure and argument to be passed to the exit procedure, which is called when the procedure which sets an exit handler using **exc_on_exit( )** exits. The exit procedures (more than one may be set) will be called regardless if the setting procedure is exited using a *return* or an **exc_raise( )**. Because the exit procedure is called as if the handling procedure had returned, the argument passed to it should not contain addresses on the handler's stack. However, any value returned by the procedure which established the exit procedure is preserved no matter what the exit procedure returns. This primitive is used in the MONITOR macro to enforce the monitor discipline on procedures.

Some signals can be considered to be synchronous traps. They are usually the starred (*) signals in the **signal**(3V) man pages. These are: **SIGSYS, SIGBUS, SIGEMT, SIGFPE, SIGILL, SIGTRAP, SIGSEGV.** If an event is marked as a trap using **agt_trap( )** (see **agt_create**(3L)) the event will generate exceptions instead of agent messages. This mapping is per-pod, not per-thread. A thread which handles the signal number of one of these as the pattern for **exc_handle( )** will catch such a signal as an exception. The exception will be raised as an **exc_notify( )** so either escape or notification style exceptions can be used, depending on what the matching **exc_handle( )** provides. If the exception is not handled, the thread will terminate. Note: it can be dangerous to supply an exception handler to treat stack overflow since the client's stack is used in raising the exception.

exc_uniqpatt( ) returns an exception pattern that is not any of the pre-defined patterns (any of the synchronous exceptions or −1 or CATCHALL). Each call to **exc_uniqpatt( )** results in a different pattern. If **exc_uniqpatt( )** cannot guarantee uniqueness, −1 is returned instead the *first* time this happens. Subsequent calls after this error result in patterns which may be duplicates.

RETURN VALUES

exc_uniqpatt( ) returns a unique pattern on success. The *first* time it fails, **exc_uniqpatt( )** returns −1.

exc_handle( ) returns:

0 on success.

−1 on failure. When **exc_handle( )** returns because of a matching call to **exc_raise( )**, it returns the *pattern* raised by **exc_raise( )**.

On success, **exc_raise( )** transfers control to the matching **exc_handle( )** and does not return. On failure, it returns −1.

exc_unhandle( ) returns:

0 on success.

−1 on failure.

exc_bound( ) returns a pointer to a function on success. On failure, it returns NULL.

On success, **exc_notify**( ) returns the return value of a function, or transfers control to a matching **exc_handle**( ) and does not return. On failure, it returns −1.

exc_on_exit( ) returns 0.

ERRORS

exc_unhandle( ) will fail if one or more of the following is true:

LE_NONEXIST	Attempt to remove a non-existent handler.
	Attempt to remove an exit handler.

exc_raise( ) will fail if one or more of the following is true:

LE_INVALIDARG	Attempt to raise an illegal pattern (−1 or 0).
LE_NONEXIST	No context found to raise an exception to.

exc_handle( ) will fail if one or more of the following is true:

LE_INVALIDARG	Attempt to handle an illegal pattern (−1 or 0).

exc_uniqpatt( ) will fail if one or more of the following is true:

LE_REUSE	Possible reuse of existing object. **agt_create**(3L), **signal**(3V)

BUGS

The stack may not contain useful information after an exception has been caught so post-exception debugging can be difficult. The reason for this is that a given handler may call procedures that trash the stack before reraising an exception.

The distinction between traps and interrupts can be problematical.

The environment restored on **exc_raise**( ) consists of the registers at the time of the **exc_handle**( ). As a result, modifications to register variables between the times of **exc_handle**( ) and **exc_raise**( ) will not be seen. This problem does not occur in the sun4 implementation.

WARNINGS

exc_on_exit( ) passes a simple type as an argument to the exit routine. If you need to pass a complex type, such as **thread_t**, **mon_t**, or **cv_t**, pass a pointer to the object instead.

NAME

lwp_create, lwp_destroy, SAMETHREAD, pod_setexit, pod_getexit, pod_exit – LWP thread creation and destruction primitives

SYNOPSIS

#include <lwp/lwp.h>
#include <lwp/stackdep.h>

int lwp_create(tid, func, prio, flags, stack, nargs, arg1, . . ., argn)
thread_t *tid;
void (*func)( );
int prio;
int flags;
stkalign_t *stack;
int nargs;
int arg1, . . . , argn;

int lwp_destroy(tid)
thread_t tid;

void pod_setexit(status)
int status;

int pod_getexit(status)
int status;

void pod_exit(status)
int status

SAMETHREAD(t1, t2)

DESCRIPTION

lwp_create( ) creates a lightweight process which starts at address *func* and has stack segment *stack*. If *stack* is NULL, the thread is created in a suspended state (see below) and no stack or pc is bound to the thread. *prio* is the scheduling priority of the thread (higher priorities are favored by the scheduler). The identity of the new thread is filled in the reference parameter *tid*. *flags* describes some options on the new thread. **LWPSUSPEND** creates the thread in suspended state (see **lwp_yield**(3L)). **LWPNOLASTRITES** will disable the **LASTRITES** agent message when the thread dies. The default (0) is to create the thread in running state with **LASTRITES** reporting enabled. **LWPSERVER** indicates that a thread is only viable as long as non-LWPSERVER threads are alive. The pod will terminate if the only living threads are marked **LWPSERVER** and blocked on a lwp resource (for instance, waiting for a message to be sent). *nargs* is the number (0 or more) of simple-type (int) arguments supplied to the thread.

The first time a lwp primitive is used, the lwp library automatically converts the caller (i.e., **main**) into a thread with the highest available scheduling priority (see **pod_getmaxpri**(3L)). The identity of this thread can be retrieved using **lwp_self** (see **lwp_status**(3L)). This thread has the normal SunOS stack given to any *forked* process.

Scheduling is, by default, non-preemptive within a priority, and within a priority, threads enter the run queue on a FIFO basis (that is, whenever a thread becomes eligible to run, it goes to the end of the run queue of its particular priority). Thus, a thread continues to run until it voluntarily relinquishes control or an event (including thread creation) occurs to enable a higher priority thread. Some primitives may cause the current thread to block, in which case the unblocked thread with the highest priority runs next. When several threads are created with the same priority, they are queued for execution in the order of creation. This order may not be preserved as threads yield and block within a priority. If an agent owned by a thread with a higher priority is invoked, that thread will preempt the currently running one.

There is no concept of ancestry in threads: the creator of a thread has no special relation to the thread it created. When all threads have died, the pod terminates.

lwp_destroy( ) is a way to explicitly terminate a thread or agent (instead of having an executing thread "fall though", which also terminates the thread). *tid* specifies the id of the thread or agent to be terminated. If *tid* is **SELF**, the invoking thread is destroyed. Upon termination, the resources (messages, monitor locks, agents) owned by the thread are released, in some cases resulting in another thread being notified of the death of its peer (by having a blocking primitive become unblocked with an error indication). A thread may terminate itself explicitly, although self-destruction is automatic when it returns from the procedure specified in the **lwp_create**( ) primitive.

pod_setexit( ) sets the exit status for a pod. This value will be returned to the parent process of the pod when the pod dies (default is 0). **exit**(3) terminates the current *thread*, using the argument supplied to *exit* to set the current value of the exit status. **on_exit**(3) establishes an action that will be taken when the entire pod terminates. **pod_exit**( ) is available to terminate the pod immediately with the final actions established by **on_exit**. If you wish to terminate the pod immediately, **pod_exit**( ) or **exit**(2V) should be used.

pod_getexit( ) returns the current value of the pod's exit status.

SAMETHREAD( ) is a convenient predicate used to compare two threads for equality.

RETURN VALUES

lwp_create( ), and **lwp_destroy**( ) return:

0	on success.
−1	on failure.

pod_getexit( ) returns the current exit status of the pod.

ERRORS

lwp_create( ) will fail if one or more of the following are true:

LE_ILLPRIO	Illegal priority.
LE_INVALIDARG	Too many arguments (> 512).
LE_NOROOM	Unable to allocate memory for thread context.

lwp_destroy( ) will fail if one or more of the following are true:

LE_NONEXIST	Attempt to destroy a thread or agent that does not exist.

SEE ALSO

exit(2V), **exit**(3), **lwp_yield**(3L), **on_exit**(3), **pod_getmaxpri**(3L)

WARNINGS

Some special threads may be created silently by the lwp library. These include an *idle* thread that runs when no other activity is going on, and a *reaper* thread that frees stacks allocated by **lwp_newstk**. These special threads will show up in status calls. A pod will terminate if these special threads are the only ones extant.

NAME

lwp_ctxinit, lwp_ctxremove, lwp_ctxset, lwp_ctxmemget, lwp_ctxmemset, lwp_fpset, lwp_libcset – special LWP context operations

SYNOPSIS

```
#include <lwp/lwp.h>

int lwp_ctxset(save, restore, ctxsize, optimize)
void (*save)(/* caddr_t ctx, thread_t old, thread_t new */);
void (*restore)(/* caddr_t ctx, thread_t old, thread_t new */);
unsigned int ctxsize;
int optimize;

int lwp_ctxinit(tid, cookie)
thread_t tid;            /* thread with special contexts */
int cookie;              /* type of context */

int lwp_ctxremove(tid, cookie)
thread_t tid;
int cookie;

int lwp_ctxmemget(mem, tid, ctx)
caddr_t mem;
thread_t tid;
int ctx;

int lwp_ctxmemset(mem, tid, ctx)
caddr_t mem;
thread_t tid;
int ctx;

int lwp_fpset(tid)
thread_t tid;            /* thread utilizing floating point hardware */

int lwp_libcset(tid)
thread_t tid;            /* thread utilizing errno */
```

DESCRIPTION

Normally on a context switch, only machine registers are saved/restored to provide each thread its own virtual machine. However, there are other hardware and software resources which can be multiplexed in this way. For example, floating point registers can be used by several threads in a pod. As another example, the global value **errno** in the standard C library may be used by all threads making system calls.

To accommodate the variety of contexts that a thread may need without requiring all threads to pay for unneeded switching overhead, **lwp_ctxinit**( ) is provided. This primitive allows a client to specify that a given thread requires certain context to be saved and restored across context switches (by default just the machine registers are switched). More than one special context may be given to a thread.

To use **lwp_ctxinit**( ), it is first necessary to define a special context. **lwp_ctxset**( ) specifies save and restore routines, as well as the size of the context that will be used to hold the switchable state. The *save* routine will automatically be invoked when an active thread is blocked and the *restore* routine will be invoked when a blocked thread is restarted. These routines will be passed a pointer to a buffer (initialized to all 0's) of size *ctxsize* which is allocated by the LWP library and used to hold the volatile state. In addition, the identity of the thread whose special context is being saved (old) and the identity of the thread being restarted (new) are passed in to the *save* and *restore* routines. **lwp_ctxset**( ) returns a cookie used by subsequent **lwp_ctxinit**( ) calls to refer to the kind of context just defined. If the *optimize* flag is TRUE, a special context switch action will not be invoked unless the thread resuming execution differs from the last thread to use the special context and also uses the special context. If the *optimize* flag is FALSE, the *save* routine will always be invoked immediately when the thread using this context is scheduled out and the *restore* routine will be invoked immediately when a new thread using this context is scheduled in. Note that an unoptimized special context is protected from threads which do not use the special context but which do affect the context state. **lwp_ctxremove( )** can be used to remove a special context installed by **lwp_ctxinit( )**.

Because context switching is done by the scheduler on behalf of a thread, it is an error to use an LWP primitive in an action done at context switch time. Also, the stack used by the save and restore routines belongs to the scheduler, so care should be taken not to use lots of stack space. As a result of these restrictions, only knowledgeable users should write their own special context switching routines.

lwp_ctxmemget( ) and **lwp_ctxmemset( )** are used to retrieve and set (respectively) the memory associated with a given special context (*ctx*) and a given thread (*tid*). *mem* is the address of client memory that will hold the context information being retrieved or set. Note that the special context *save* and *restore* routines may be NULL, so pure data may be associated with a given thread using these primitives.

Several kinds of special contexts are predefined. To allow a thread to share floating point hardware with other threads, the **lwp_fpset( )** primitive is available. The floating-point hardware bound at compile-time is selected automatically. To multiplex the global variable **errno**, **lwp_libcset( )** is used to have **errno** become part of the context of thread *tid*.

Special contexts can be used to assist in managing stacks. See **lwp_newstk**(3L) for details.

RETURN VALUES

On success, **lwp_ctxset( )** returns a cookie to be used by subsequent calls to **lwp_ctxinit( )**. If unable to define the context, it returns −1.

ERRORS

lwp_ctxinit( ) will fail if one or more of the following are true:

LE_INUSE	This special context already set for this thread.

lwp_ctxremove( ) will fail if one or more of the following are true:

LE_NONEXIST	The specified context is not set for this thread.

lwp_ctxset( ) will fail if one or more of the following are true:

LE_NOROOM	Unable to allocate memory to define special context.

SEE ALSO

lwp_newstk(3L)

BUGS

The floating point contexts should be initialized implicitly for those threads that use floating point.

NAME

lwp_checkstkset, lwp_stkcswset, CHECK, lwp_setstkcache, lwp_newstk, lwp_datastk, STKTOP – LWP stack management

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/check.h>
#include <lwp/lwpmachdep.h>
#include <lwp/stackdep.h>

CHECK(location, result)

int lwp_checkstkset(tid, limit)
thread_t tid;
caddr_t limit;

int lwp_stkcswset(tid, limit)
thread_t tid;
caddr_t limit;

int lwp_setstkcache(minstksz, numstks)
int minstksz;
int numstks;

stkalign_t *lwp_newstk()

stkalign_t *lwp_datastk(data, size, addr)
caddr_t data;
int size;
caddr_t *addr;

STKTOP(s)
```

DESCRIPTION

Stacks are problematical with lightweight processes. What is desired is that stacks for each thread are red-zone protected so that one thread's stack does not unexpectedly grow into the stack of another. In addition, stacks should be of infinite length, grown as needed. The process stack is a maximum-sized segment (see **getrlimit**(2).) This stack is redzone protected, and you can even try to extend it beyond its initial maximum size in some cases. With SunOS 4.*x*, it is possible to efficiently allocate large stacks that have red zone protection, and the LWP library provides some support for this. For those systems that do not have flexible memory management, the LWP library provides assistance in dealing with the problems of maintaining multiple stacks.

The stack used by **main**( ) is the same stack that the system allocates for a process on **fork**(2V). For allocating other thread stacks, the client is free to use any statically or dynamically allocated memory (using memory from **main**( )'s stack is subject to the stack resource limit for any process created by **fork**( )). In addition, the **LASTRITES** agent message is available to free allocated resources when a thread dies. The size of any stack should be at least **MINSTACKSZ * sizeof (stkalign_t)**, because the LWP library will use the client stack to execute primitives. For very fast dynamically allocated stacks, a stack cacheing mechanism is available. **lwp_setstkcache**( ) allocates a cache of stacks. Each time the cache is empty, it is filled with *numstks* new stacks, each containing at least *minstksz* bytes. *minstksz* will automatically be augmented to take into account the stack needs of the LWP library. **lwp_newstk**( ) returns a cached stack that is suitable for use in an **lwp_create**( ) call. **lwp_setstkcache**( ) must be called (once) prior to any use of **lwp_newstk**. If running under SunOS 4.*x*, the stacks allocated by **lwp_newstk**( ) will be red-zone protected (an attempt to reference below the stack bottom will result in a **SIGSEGV** event).

Threads created with stacks from **lwp_newstk**( ) should not use the **NOLASTRITES** flag. If they do, cached stacks will not be returned to the cache when a thread dies.

lwp_datastk( ) also returns a red-zone protected stack like **lwp_newstk( )** does. It copies any amount of data (subject to the size limitations imposed by **lwp_setstkcache**) onto the stack *above* the stack top that it returns. *data* points to information of *size* bytes to be copied. The exact location where the data is stored is returned in the reference parameter *addr*. Because **lwp_create( )** only passes simple types to the newly-created thread, **lwp_datastk( )** is useful to pass a more complex argument: Call **lwp_datastk( )** to get an initialized stack, and pass the address of the data structure (*addr*) as an argument to the new thread.

A *reaper* thread running at the maximum pod priority is created by **lwp_setstkcache**. It's action may be delayed by other threads running at that priority, so it is suggested that the maximum pod priority not be used for client-created threads when **lwp_newstk( )** is being used. Altering the maximum pod priority with **pod_setmaxpri( )** will have the side effect of increasing the reaper thread priority as well.

The stack address passed to **lwp_create( )** represents the top of the stack: the LWP library will not use any addresses at or above it. Thus, it is safe to store information above the stack top if there is room there.

For stacks that are not protected with hardware redzones, some protection is still possible. For any thread *tid* with stack boundary *limit* made part of a special context with **lwp_checkstkset( )**, the **CHECK** macro may be used. This macro, if used at the beginning of each procedure (and before local storage is initialized (it is all right to *declare* locals though)), will check that the stack limit has not been violated. If it has, the non-local *location* will be set to *result* and the procedure will return. **CHECK** is not perfect, as it is possible to call a procedure with many arguments after **CHECK** validates the stack, only to have these arguments clobber the stack before the new procedure is entered.

lwp_stkcswset( ) checks at context-switch time the stack belonging to thread *tid* for passing stack boundary *limit*. In addition, a checksum at the bottom of the stack is validated to ensure that the stack did not temporarily grow beyond its limit. This is automated and more efficient than using **CHECK**, but by the time a context switch occurs, it's too late to do much but **abort**(3) if the stack was clobbered.

To portably use statically allocated stacks, the macros in **<lwp/stackdep.h>** should be used. Declare a stack *s* to be an array of **stkalign_t**, and pass the stack to **lwp_create( )** as STKTOP(s).

RETURN VALUES

lwp_checkstkset( ) and **lwp_stkcswset( )** return 0.

lwp_setstkcache( ) returns the actual size of the stacks allocated in the cache.

lwp_newstk( ) and **lwp_datastk( )** return a valid new stack address on success. On failure, they return 0.

SEE ALSO

getrlimit(2), **abort**(3)

WARNINGS

lwp_datastk( ) should not be directly used in a **lwp_create( )** call since C does not guarantee the order in which arguments to a function are evaluated.

BUGS

C should provide support for heap-allocated stacks at procedure entry time. The hardware should be segment-based to eliminate the problem altogether.

NAME

lwp_geterr, lwp_perror, lwp_errstr – LWP error handling

SYNOPSIS

#include <lwp/lwp.h>
#include <lwp/lwperror.h>

lwp_err_t lwp_geterr( );

void
lwp_perror(s)
char *s;

char **lwp_errstr( );

DESCRIPTION

When a primitive fails (returns −1), **lwp_geterr( )** can be used to obtain the identity of the error (which is part of the context for each lwp). **lwp_perror( )** can be used to print an error message on the standard error file (analogous to **perror**(3)) when a lwp primitive returns an error indication. **lwp_perror( )** uses the same mechanism as **lwp_geterr( )** to obtain the last error. **lwp_errstr** returns a pointer to the (NULL-terminated) list of error messages.

lwp_libcset (see **lwp_ctxinit**(3L)) allows **errno** from the standard C library reflect a per-thread value rather than a per-pod value.

SEE ALSO

lwp_ctxinit(3L), **perror**(3)

NAME

lwp_self, lwp_ping, lwp_enumerate, lwp_getstate, lwp_setregs, lwp_getregs – LWP status information

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/lwpmachdep.h>

int
lwp_enumerate(vec, maxsize)
thread_t vec[ ];  /* list of id's to be filled in */
int maxsize;      /* number of elements in vec */

int
lwp_ping(tid)
thread_t tid;

int
lwp_getregs(tid, machstate)
thread_t tid;
machstate_t *machstate;

int
lwp_setregs(tid, machstate)
thread_t tid;
machstate_t *machstate;

int
lwp_getstate(tid, statvec)
thread_t tid;
statvec_t *statvec;

int
lwp_self(tid)
thread_t *tid;
```

DESCRIPTION

lwp_self( ) returns the ID of the current thread in *tid*. This is the *only* way to retrieve the identity of *main*.

lwp_enumerate( ) fills in a list with the ID's of all existing threads and returns the total number of threads. This primitive will use *maxsize* to avoid exceeding the capacity of the list. If the number of threads is greater than *maxsize*, only *maxsize* thread ID's are filled in *vec*. If *maxsize* is zero, **lwp_enumerate( )** just returns the total number of threads.

lwp_getstate( ) is used to retrieve the context of a given thread. It is possible to see what object (thread, monitor, etc.) if any that thread is blocked on, and the scheduling priority of the thread.

lwp_ping returns 0 (no error) if the thread *tid* exists. Otherwise, -1 is returned.

lwp_setregs sets the machine-dependent context (i.e., registers) of a thread. The next time the thread is scheduled in, this context is installed. Consult **lwpmachdep.h** for the details. **lwp_getregs** retrieves the machine-dependent context. Note: the registers may not be meaningful unless the thread in question is blocked or suspended because the state of the registers as of the most recent context switch is returned.

RETURNS

Upon successful completion, **lwp_self** and **lwp_getstate( )** return 0, –1 on error.

lwp_enumerate( ) returns the total number of threads.

lwp_ping returns 0 if the specified thread exists, else -1.

ERRORS

lwp_getstatea( ) , **lwp_ping( )** , and **lwp_setstate( )** will fail if one or more of the following is true:

LE_NONEXIST	Attempt to get the status of a non-existent thread.

NAME

lwp_yield, lwp_suspend, lwp_resume, lwp_join, lwp_setpri, lwp_resched, lwp_sleep – control LWP scheduling

SYNOPSIS

#include <lwp/lwp.h>

int lwp_yield(tid)
thread_t tid;

int lwp_sleep(timeout)
struct timeval *timeout;

int lwp_resched(prio)
int prio;

int lwp_setpri(tid, prio)
thread_t tid;
int prio;

int lwp_suspend(tid)
thread_t tid;

int lwp_resume(tid)
thread_t tid;

int lwp_join(tid)
thread_t tid;

DESCRIPTION

lwp_yield( ) allows the currently running thread to voluntarily relinquish control to another thread *with the same scheduling priority.* If *tid* is **SELF**, the next thread in the same priority queue of the yielding thread will run and the current thread will go the end of the scheduling queue. Otherwise, it is the ID of the thread to run next, and the current thread will take second place in the scheduling queue.

lwp_sleep( ) blocks the thread executing this primitive for at least the time specified by *timeout*.

Scheduling of threads is, by default, preemptive (higher priorities preempt lower ones) across priorities and non-preemptive within a priority. **lwp_resched( )** moves the front thread for a given priority to the end of the scheduling queue. Thus, to achieve a preemptive round–robin scheduling discipline, a high priority thread can periodically wake up and shuffle the queue of threads at a lower priority. **lwp_resched( )** does not affect threads which are blocked. If the priority of the rescheduled thread is the same as that of the caller, the effect is the same as **lwp_yield( )**.

lwp_setpri( ) is used to alter (raise or lower) the scheduling priority of the specified thread. If *tid* is **SELF**, the priority of the invoking thread is set. Note: if the priority of the affected thread becomes greater than that of the caller and the affected thread is not blocked, the caller will not run next. **lwp_setpri( )** can be used on either blocked or unblocked threads.

lwp_join( ) blocks the thread issuing the join until the thread *tid* terminates. More than one thread may join *tid*.

lwp_suspend( ) makes the specified thread ineligible to run. If *tid* is **SELF**, the caller is itself suspended. **lwp_resume( )** undoes the effect of **lwp_suspend( )**. If a blocked thread is suspended, it will not run until it has been unblocked as well as explicitly made eligible to run using **lwp_resume( )**. By suspending a thread, one can safely examine it without worrying that its execution–time state will change.

NOTES

When scheduling preemptively, be sure to use monitors to protect shared data structures such as those used by the standard I/O library.

RETURN VALUES

lwp_yield(), **lwp_sleep()**, **lwp_resched()**, **lwp_join()**, **lwp_suspend()** and **lwp_resume()** return:

0	on success.
−1	on failure.

lwp_setpri() returns the previous priority on success. On failure, it returns −1.

ERRORS

lwp_yield() will fail if one or more of the following is true:

LE_ILLPRIO	Attempt to yield to thread with different priority.
LE_INVALIDARG	Attempt to yield to a blocked thread.
LE_NONEXIST	Attempt to yield to a non-existent thread.

lwp_sleep() will fail if one or more of the following is true:

LE_INVALIDARG	Illegal timeout specified.

lwp_resched() will fail if one or more of the following is true:

LE_ILLPRIO	The priority queue specified contains no threads to reschedule.
LE_INVALIDARG	Attempt to reschedule thread at priority greater than that of the caller.

lwp_setpri() will fail if one or more of the following is true:

LE_INVALIDARG	The priority specified is beyond the maximum available to the pod.
LE_NONEXIST	Attempt to set priority of a non-existent thread.

lwp_join() will fail if one or more of the following are true:

LE_NONEXIST	Attempt to join a thread that does not exist.

lwp_suspend() will fail if one or more of the following is true:

LE_NONEXIST	Attempt to suspend a non-existent thread.

lwp_resume() will fail if one or more of the following is true:

LE_NONEXIST	Attempt to resume a non-existent thread.

NAME

mon_create, mon_destroy, mon_enter, mon_exit, mon_enumerate, mon_waiters, mon_cond_enter, mon_break, MONITOR, SAMEMON – LWP routines to manage critical sections

SYNOPSIS

```
#include <lwp/lwp.h>

int mon_create(mid)
mon_t *mid;

int mon_destroy(mid)
mon_t mid;

int mon_enter(mid)
mon_t mid;

int mon_exit(mid)
mon_t mid;

int mon_enumerate(vec, maxsize)
mon_t vec[ ];      /* list of all monitors */
int maxsize;       /* max size of vec */

int mon_waiters(mid, owner, vec, maxsize)
mon_t mid;               /* monitor in question */
thread_t *owner;         /* which thread owns the monitor */
thread_t vec[ ];         /* list of blocked threads */
int maxsize;             /* max size of vec */

int mon_cond_enter(mid)
mon_t mid;

int mon_break(mid)
mon_t mid;

void MONITOR(mid)
mon_t mid;

int SAMEMON(m1, m2)
mon_t m1;
mon_t m2;
```

DESCRIPTION

Monitors are used to synchronize access to common resources. Although it is possible (on a uniprocessor) to use knowledge of how scheduling priorities work to serialize access to a resource, monitors (and condition variables) provide a general tool to provide the necessary synchronization.

mon_create( ) creates a new monitor and returns its identity in *mid*. **mon_destroy( )** destroys a monitor, as well as any conditions bound to it (see **cv_create**(3L)). Because the lifetime of a monitor can transcend the lifetime of the LWP that created it, monitor destruction is not automatic upon LWP destruction.

mon_enter( ) blocks the calling thread (if the monitor is in use) until the monitor becomes free by being exited or by waiting on a condition (see **cv_create**(3L)). Threads unable to gain entry into the monitor are queued for monitor service by the priority of the thread requesting monitor access, FCFS within a priority. Monitor calls may nest. If, while holding monitor M1 a request for monitor M2 is made, M1 will be held until M2 can be acquired.

mon_cond_enter( ) will enter the monitor only if the monitor is not busy. Otherwise, an error is returned.

mon_enter( ) and **mon_cond_enter( )** will allow a thread which already has the monitor to reenter the monitor. In this case, the nesting level of monitor entries is returned. Thus, the first time a monitor is entered, **mon_enter( )** returns 0. The next time the monitor is entered, **mon_enter( )** returns 1. **mon_exit( )** frees the current monitor and allows the next thread blocked on the monitor (if any) to enter the monitor. However, if a monitor is entered more than once, **mon_exit( )** returns the previous monitor nesting level without freeing the monitor to other threads. Thus, if the monitor was not reentered, **mon_exit( )** returns 0.

mon_enumerate( ) lists all the monitors in the system. The vector supplied is filled in with the ID's of the monitors. *maxsize* is used to avoid exceeding the capacity of the list. If the number of monitors is greater than *maxsize*, only *maxsize* monitor ID's are filled in *vec*.

mon_waiters( ) puts the thread that currently owns the monitor in *owner* and all threads blocked on the monitor in *vec* (subject to the *maxsize* limitation), and returns the number of waiting threads.

mon_break( ) forces the release of a monitor lock not necessarily held by the invoking thread. This enables the next thread blocked on the monitor to enter it.

MONITOR is a macro that can be used at the start of a procedure to indicate that the procedure is a monitor. It uses the exception handling mechanism to ensure that the monitor is exited automatically when the procedure exits. Ordinarily, this single macro replaces paired **mon_enter( )**- **mon_exit( )** calls in a monitor procedure.

The **SAMEMON** macro is a convenient predicate used to compare two monitors for equality.

Monitor locks are released automatically when the LWP holding them dies. This may have implications for the validity of the monitor invariant (a condition that is always true *outside* of the monitor) if a thread unexpectedly terminates.

RETURN VALUES

mon_create( ) returns the ID of a new monitor.

mon_destroy( ) returns:

0 on success.

−1 on failure.

mon_enter( ) returns the nesting level of the monitor.

mon_exit( ) returns the previous nesting level on success. On failure, it returns −1.

mon_enumerate( ) returns the total number of monitors.

mon_waiters( ) returns the number of threads waiting for the monitor.

mon_cond_enter( ) returns the nesting level of the monitor if the monitor is not busy. If the monitor is busy, it returns −1.

mon_break( ) returns:

0 on success.

−1 on failure.

The macro **SAMEMON( )** returns 1 if the monitors specified by *m1* and *m2* are equal. It returns 0 otherwise.

ERRORS

mon_break( ) will fail if one or more of the following are true:

LE_NONEXIST	Attempt to break lock on non-existent monitor.
LE_NOTOWNED	Attempt to break a monitor lock that is not set.

mon_cond_enter( ) will fail if one or more of the following are true:

LE_INUSE	The requested monitor is being used by another thread.
LE_NONEXIST	Attempt to destroy non-existent monitor.

mon_destroy( ) will fail if one or more of the following are true:

LE_INUSE	Attempt to destroy a monitor that has threads blocked on it.
LE_NONEXIST	Attempt to destroy non-existent monitor.

mon_exit( ) will fail if one or more of the following are true:

LE_INVALIDARG	Attempt to exit a monitor that the thread does not own.
LE_NONEXIST	Attempt to exit non-existent monitor.

SEE ALSO

cv_create(3L)

BUGS

There should be language support to enforce the monitor enter-exit discipline.

NAME

msg_send, msg_recv, msg_reply, MSG_RECVALL, msg_enumsend, msg_enumrecv – LWP send and receive messages

SYNOPSIS

```
#include <lwp/lwp.h>

int msg_send(dest, arg, argsize, res, ressize)
thread_t dest;     /* destination thread */
caddr_t arg;       /* argument buffer */
int argsize;       /* size of argument buffer */
caddr_t res;       /* result buffer */
int ressize;       /* size of result buffer */

int msg_recv(sender, arg, argsize, res, ressize, timeout)
thread_t *sender;          /* value-result: sending thread or agent */
caddr_t *arg;              /* argument buffer */
int *argsize;              /* argument size */
caddr_t *res;              /* result buffer */
int *ressize;              /* result size */
struct timeval *timeout; /* POLL, INFINITY, else timeout */

int msg_reply(sender)
thread_t sender;/* agent id or thread id */

int msg_enumsend(vec, maxsize)
thread_t vec[ ];  /* list of blocked senders */
int maxsize;

int msg_enumrecv(vec, maxsize)
thread_t vec[ ];  /* list of blocked receivers */
int maxsize;

int MSG_RECVALL(sender, arg, argsize, res, ressize, timeout)
thread_t *sender;
caddr_t *arg;
int *argsize;
caddr_t *res;
int *ressize;
struct timeval *timeout;
```

DESCRIPTION

Each thread queues messages addressed to it as they arrive. Threads may either specify that a particular sender's message is to be received next, or that *any* sender's message may be received next.

msg_send( ) specifies a message buffer and a reply buffer, and initiates one half of a rendezvous with the receiver. The sender will block until the receiver replies using **msg_reply( )**. **msg_recv( )** initiates the other half of a rendezvous and blocks the invoking thread until a corresponding **msg_send( )** is received. When unblocked by **msg_send( )**, the receiver may read the message and generate a reply by filling in the reply buffer and issuing **msg_reply( )**. **msg_reply( )** unblocks the sender. Once a reply is sent, the receiver should no longer access either the message or reply buffer.

In **msg_send( )**, *argsize* specifies the size in bytes of the argument buffer *argbuf*, which is intended to be a read-only (to the receiver) buffer. *ressize* specifies the size in bytes of the result buffer *resbuf*, which is intended to be a write-only (to the receiver) buffer. *dest* is the thread that is the target of the send.

msg_recv( ) blocks the receiver until:

- A message from the agent or thread bound to *sender* has been sent to the receiver or,
- *sender* points to a THREADNULL-valued variable and *any* message has been sent to the receiver from a thread or agent, or,
- After the time specified by *timeout* elapses and no message is received.

If *timeout* is **POLL**, **msg_recv( )** returns immediately, returning success if the message expected has arrived; otherwise an error is returned. If *timeout* is **INFINITY**, **msg_recv( )** blocks forever or until the expected message arrives. If *timeout* is any other value **msg_recv( )** blocks for the time specified by *timeout* or until the expected message arrives, whichever comes first. When **msg_recv( )** returns, *sender* is filled in with the identity of the sending thread or agent, and the buffer addresses and sizes specified by the matching send are stored in *arg*, *argsize*, *res*, and *ressize*.

msg_enumsend( ) and **msg_enumrecv( )** are used to list all of the threads blocked on sends (awaiting a reply) and receives (awaiting a send), respectively. The value returned is the number of such blocked threads. The vector supplied by the client is filled in (subject to the *maxsize* limitation) with the ID's of the blocked threads. *maxsize* is used to avoid exceeding the capacity of the list. If the number of threads blocked on sends or receives is greater than *maxsize*, only *maxsize* thread ID's are filled in *vec*. If *maxsize* is 0, just the total number of blocked threads is returned.

sender in **msg_recv( )** is a reference parameter. If you wish to receive from *any* sender, be sure to reinitialize the thread *sender* points to as THREADNULL before each use (do not use the address of THREADNULL for the sender). Alternatively, use the **MSG_RECVALL( )** macro. This macro has the same parameters as **msg_recv( )**, but ensures that the sender is properly initialized to allow receipt from any sender. **MSG_RECVALL( )** returns the result from **msg_recv**.

RETURN VALUES

msg_send( ), **msg_recv( )**, **MSG_RECVALL( )** and **msg_reply( )** return:

0	on success.
−1	on failure.

msg_enumsend( ) returns the number of threads blocked on **msg_send( )**.

msg_enumrecv( ) returns the number of threads blocked on **msg_recv( )**.

ERRORS

msg_recv( ) will fail if one or more of the following is true:

LE_INVALIDARG	An illegal timeout was specified.
	The sender address is that of THREADNULL.
LE_NONEXIST	The specified thread or agent does not exist.
LE_TIMEOUT	Timed out before message arrived.

msg_reply( ) will fail if one or more of the following is true:

LE_NONEXIST	Attempt to reply to a sender that does not exist or has terminated.
LE_NOWAIT	Attempt to reply to a sender that is not expecting a reply.

msg_send( ) will fail if one or more of the following is true:

LE_INVALIDARG	Attempt to send a message to yourself.
LE_NONEXIST	The specified destination thread does not exist or has terminated.

NAME

pod_getmaxpri, pod_getmaxsize, pod_setmaxpri – control LWP scheduling priority

SYNOPSIS

int pod_getmaxpri( )

int pod_getmaxsize( )

int pod_setmaxpri(maxprio)
int maxprio;

DESCRIPTION

The LWP library is self-initializing: the first time you use a primitive that requires threads to be supported, *main* is automatically converted into a thread. A pod will terminate when all client-created lightweight threads (including the thread bound to *main*) are dead.

By default, only a single priority (**MINPRIO**) is available. However, by using **pod_setmaxpri( )**, you can make an arbitrary number (up to the limit imposed by the implementation) of priorities available. The *main* thread will receive the highest available scheduling priority at the time of initialization. By using **pod_setmaxpri( )** before any other LWP primitives, you can ensure that main will receive the same priority as the argument to **pod_setmaxpri( )**. **pod_setmaxpri( )** can be called repeatedly, as long as the number of scheduling priorities (*maxprio*) increases with each call.

pod_getmaxpri( ) returns the current number of available priorities. Priorities are numbered from 1 (**MINPRIO**) to **MAXPRIO**.

The implementation-dependent maximum number of priorities available can be retrieved using **pod_getmaxsize( )**. This value will never be less than 255.

RETURN VALUES

pod_getmaxpri( ) returns the number of priority levels set by the most recent call to **pod_setmaxpri( )**.

pod_getmaxsize( ) returns the maximum number of priorities your system supports.

pod_setmaxpri( ) returns:

0	on success.
–1	on failure.

ERRORS

pod_setmaxpri( ) will fail if one or more of the following are true:

LE_INVALIDARG	Attempt to allocate more priorities than supported.
LE_NOROOM	No internal memory left to create pod.

NAME

intro – introduction to mathematical library functions and constants

SYNOPSIS

#include <sys/ieeefp.h>

#include <floatingpoint.h>

#include <math.h>

DESCRIPTION

The include file **<math.h>** contains declarations of all the functions described in Section 3M that are implemented in the math library, **libm**. C programs should be linked with the **–lm** option in order to use this library.

<sys/ieeefp.h> and **<floatingpoint.h>** define certain types and constants used for **libm** exception handling, conforming to ANSI/IEEE Std 754-1985, the *IEEE Standard for Binary Floating-Point Arithmetic*.

ACKNOWLEDGEMENT

The Sun version of **libm** is based upon and developed from ideas embodied and codes contained in 4.3 BSD, which may not be compatible with earlier BSD or UNIX implementations.

IEEE ENVIRONMENT

The IEEE Standard specifies modes for rounding direction, precision, and exception trapping, and status reflecting accrued exceptions. These modes and status constitute the IEEE run-time environment. On Sun-2 and Sun-3 systems without 68881 floating-point co-processors, only the default rounding direction to nearest is available, only the default non-stop exception handling is available, and accrued exception bits are not maintained.

IEEE EXCEPTION HANDLING

The IEEE Standard specifies exception handling for **aint**, **ceil**, **floor**, **irint**, **remainder**, **rint**, and **sqrt**, and suggests appropriate exception handling for **fp_class**, **copysign**, **fabs**, **finite**, **fmod**, **isinf**, **isnan**, **ilogb**, **ldexp**, **logb**, **nextafter**, **scalb**, **scalbn** and **signbit**, but does not specify exception handling for the other **libm** functions.

For these other unspecified functions the spirit of the IEEE Standard is generally followed in **libm** by handling invalid operand, singularity (division by zero), overflow, and underflow exceptions, as much as possible, in the same way they are handled for the fundamental floating-point operations such as addition and multiplication.

These unspecified functions are usually not quite correctly rounded, may not observe the optional rounding directions, and may not set the inexact exception correctly.

SYSTEM V EXCEPTION HANDLING

The *System V Interface Definition* (SVID) specifies exception handling for some **libm** functions: **j0( )**, **j1( )**, **jn( )**, **y0( )**, **y1( )**, **yn( )**, **exp( )**, **log( )**, **log10( )**, **pow( )**, **sqrt( )**, **hypot( )**, **lgamma( )**, **sinh( )**, **cosh( )**, **sin( )**, **cos( )**, **tan( )**, **asin( )**, **acos( )**, and **atan2( )**. See **matherr**(3M) for a discussion of the extent to which Sun's implementation of **libm** follows the SVID when it is consistent with the IEEE Standard and with hardware efficiency.

LIST OF MATH LIBRARY FUNCTIONS

Name	Appears on Page	Description
–	**bessel**(3M)	Bessel functions
–	**frexp**(3M)	floating-point analysis
–	**hyperbolic**(3M)	hyperbolic functions
–	**ieee_functions**(3M)	IEEE classification
–	**ieee_test**(3M)	IEEE tests for compliance
–	**ieee_values**(3M)	returns double-precision IEEE infinity
–	**trig**(3M)	trigonometric functions
acos	**trig**(3M)	trigonometric functions

acosh	**hyperbolic**(3M)	hyperbolic functions
aint	**rint**(3M)	round to integral value in floating-point or integer format
anint	**rint**(3M)	round to integral value in floating-point or integer format
annuity	**exp**(3M)	exponential, logarithm, power
asin	**trig**(3M)	trigonometric functions
asinh	**hyperbolic**(3M)	hyperbolic functions
atan	**trig**(3M)	trigonometric functions
atan2	**trig**(3M)	trigonometric functions
atanh	**hyperbolic**(3M)	hyperbolic functions
cbrt	**sqrt**(3M)	cube root, square root
ceil	**rint**(3M)	round to integral value in floating-point or integer format
compound	**exp**(3M)	exponential, logarithm, power
copysign	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
cos	**trig**(3M)	trigonometric functions
cosh	**hyperbolic**(3M)	hyperbolic functions
erf	**erf**(3M)	error functions
erfc	**erf**(3M)	error functions
exp	**exp**(3M)	exponential, logarithm, power
exp2	**exp**(3M)	exponential, logarithm, power
exp10	**exp**(3M)	exponential, logarithm, power
expm1	**exp**(3M)	exponential, logarithm, power
fabs	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
finite	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
floor	**rint**(3M)	round to integral value in floating-point or integer format
fmod	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
fp_class	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
frexp	**frexp**(3M)	traditional UNIX functions
HUGE	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
HUGE_VAL	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
hypot	**hypot**(3M)	Euclidean distance
ieee_flags	**ieee_flags**(3M)	mode and status function for IEEE standard arithmetic
ieee_functions	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
ieee_handler	**ieee_handler**(3M)	IEEE exception trap handler function
ieee_test	**ieee_test**(3M)	IEEE test functions for verifying standard compliance
ieee_values	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
ilogb	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
infinity	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
irint	**rint**(3M)	round to integral value in floating-point or integer format
isinf	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
isnan	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
isnormal	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
issubnormal	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
iszero	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
j0	**bessel**(3M)	Bessel functions
j1	**bessel**(3M)	Bessel functions
jn	**bessel**(3M)	Bessel functions
ldexp	**frexp**(3M)	traditional UNIX functions
lgamma	**lgamma**(3M)	log gamma function
log	**exp**(3M)	exponential, logarithm, power
log2	**exp**(3M)	exponential, logarithm, power
log10	**exp**(3M)	exponential, logarithm, power
log1p	**exp**(3M)	exponential, logarithm, power
logb	**ieee_test**(3M)	IEEE test functions for verifying standard compliance

matherr	**matherr**(3M)	math library exception-handling function
max_normal	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
max_subnormal	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
min_normal	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
min_subnormal	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
modf	**frexp**(3M)	traditional UNIX functions
nextafter	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
nint	**rint**(3M)	round to integral value in floating-point or integer format
pow	**exp**(3M)	exponential, logarithm, power
quiet_nan	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
remainder	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
rint	**rint**(3M)	round to integral value in floating-point or integer format
scalb	**ieee_test**(3M)	IEEE test functions for verifying standard compliance
scalbn	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
signaling_nan	**ieee_values**(3M)	functions that return extreme values of IEEE arithmetic
signbit	**ieee_functions**(3M)	miscellaneous functions for IEEE arithmetic
significant	**ieee_test**(3M)	IEEE test functions for verifying standard compliance
sin	**trig**(3M)	trigonometric functions
single_precision	**single_precision**(3M)	single-precision access to libm functions
sinh	**hyperbolic**(3M)	hyperbolic functions
sqrt	**sqrt**(3M)	cube root, square root
tan	**trig**(3M)	trigonometric functions
tanh	**hyperbolic**(3M)	hyperbolic functions
y0	**bessel**(3M)	Bessel functions
y1	**bessel**(3M)	Bessel functions
yn	**bessel**(3M)	Bessel functions

NAME

j0, j1, jn, y0, y1, yn – Bessel functions

SYNOPSIS

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x)
double x;
int n;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
int n;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

SEE ALSO

exp(3M)

DIAGNOSTICS

The functions *y0, y1,* and *yn* have logarithmic singularities at the origin, so they treat zero and negative arguments the way *log* does, as described in **exp**(3M). Such arguments are unexceptional for *j0, j1,* and *jn.*

NAME

erf, erfc – error functions

SYNOPSIS

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

DESCRIPTION

erf(x) returns the error function of x; where $\mathbf{erf}(x) := (2/\sqrt{\pi}) \int_0^x \exp(-t^2)\,dt$.

erfc(x) returns 1.0–erf (x), computed however by other methods that avoid cancellation for large x.

NAME

exp, expm1, exp2, exp10, log, log1p, log2, log10, pow, compound, annuity – exponential, logarithm, power

SYNOPSIS

#include <math.h>

double exp(x)
double x;

double expm1(x)
double x;

double exp2(x)
double x;

double exp10(x)
double x;

double log(x)
double x;

double log1p(x)
double x;

double log2(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double compound(r, n)
double r, n;

double annuity(r, n)
double r, n;

DESCRIPTION

exp() returns the exponential function *e**x*.

expm1() returns *e**x–1* accurately even for tiny *x*.

exp2() and **exp10()** return *2**x* and *10**x* respectively.

log() returns the natural logarithm of *x*.

log1p() returns log(1+x) accurately even for tiny *x*.

log2() and **log10()** return the logarithm to base 2 and 10 respectively.

pow() returns *x**y*. **pow(***x* **,0.0)** is 1 for all x, in conformance with 4.3BSD, as discussed in the *Numerical Computation Guide*.

compound() and **annuity()** are functions important in financial computations of the effect of interest at periodic rate *r* over *n* periods. **compound(***r*, *n***)** computes *(1+r)**n*, the compound interest factor. Given an initial principal *P0,* its value after *n* periods is just **Pn** = *P0* * **compound(***r*, *n***)**. **annuity(***r*, *n***)** computes *(1 – (1+r)**–n)/r,* the present value of annuity factor. Given an initial principal *P0,* the equivalent periodic payment is just **p** = *P0* / **annuity(***r*, *n***)**. **compound()** and **annuity()** are computed using **log1p()** and **expm1()** to avoid gratuitous inaccuracy for small-magnitude *r*. **compound()** and **annuity()** are not defined for *r* <= –1.

Thus a principal amount *P0* placed at 5% annual interest compounded quarterly for 30 years would yield

P30 = *P0* *** compound(.05/4, 30.0 * 4)**

while a conventional fixed-rate 30-year home loan of amount *P0* at 10% annual interest would be amortized by monthly payments in the amount

p = *P0* **/ annuity(.10/12, 30.0 * 12)**

SEE ALSO

matherr(3M)

DIAGNOSTICS

All these functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. Thus for x == ±0, **log**(*x*) is $-\infty$ with a division by zero exception; for x < 0, including $-\infty$, **log**(*x*) is a quiet NaN with an invalid operation exception; for x == $+\infty$ or a quiet NaN, **log**(*x*) is x without exception; for x a signaling NaN, **log**(*x*) is a quiet NaN with an invalid operation exception; for x == 1, **log**(*x*) is 0 without exception; for any other positive x, **log**(*x*) is a normalized number with an inexact exception.

In addition, **exp**( ), **exp2**( ), **exp10**( ), **log**( ), **log2**( ), **log10**( ) and **pow**( ) may also set **errno** and call **matherr**(3M).

NAME

frexp, modf, ldexp – traditional UNIX functions

SYNOPSIS

```
#include <math.h>

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(x,n)
double x;
int n;

double modf(value, iptr)
double value, *iptr;
```

DESCRIPTION

These functions are provided for compatibility with other UNIX system implementations. They are not used internally in **libm** or **libc**. Better ways to accomplish similar ends may be found in **ieee_functions**(3M) and **rint**(3M).

ldexp(*x*,*n*) returns *x* * *2***n* computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Note: **ldexp**(*x*,*n*) differs from **scalbn**(*x*,*n*), defined in **ieee_functions**(3M), only that in the event of IEEE overflow and underflow, **ldexp**(*x*,*n*) sets **errno** to ERANGE.

Every non-zero number can be written uniquely as *x* * *2***n*, where the significant *x* is in the range *0.5* <= |*x*| < *1.0* and the exponent *n* is an integer. The function **frexp**( ) returns the significant of a double *value* as a double quantity, *x*, and stores the exponent *n*, indirectly through *eptr*. If *value* == 0, both results returned by **frexp**( ) are 0.

modf( ) returns the fractional part of *value* and stores the integral part indirectly through *iptr*. Thus the argument *value* and the returned values **modf**( ) and **iptr* satisfy

(**iptr* + *modf*) == *value*

and both results have the same sign as *value*. The definition of **modf**( ) varies among UNIX system implementations, so avoid **modf**( ) in portable code.

The results of **frexp**( ) and **modf**( ) are not defined when *value* is an IEEE infinity or NaN.

SEE ALSO

ieee_functions(3M), **rint**(3M)

NAME

sinh, cosh, tanh, asinh, acosh, atanh – hyperbolic functions

SYNOPSIS

#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;

double asinh(x)
double x;

double acosh(x)
double x;

double atanh(x)
double x;

DESCRIPTION

These functions compute the designated direct and inverse hyperbolic functions for real arguments. They inherit much of their roundoff error from **expm1**( ) and **log1p**, described in **exp**(3M).

DIAGNOSTICS

These functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. Thus **sinh**( ) and **cosh**( ) return $\pm\infty$ on overflow, **acosh**( ) returns a NaN if its argument is less than 1, and **atanh**( ) returns a NaN if its argument has absolute value greater than 1. In addition, **sinh**,**cosh**, and **tanh**( ) may also set **errno** and call **matherr**(3M).

SEE ALSO

exp(3M), **matherr**(3M)

NAME

hypot – Euclidean distance

SYNOPSIS

#include <math.h>

double hypot(x, y)
double x, y;

DESCRIPTION

hypot( ) returns

sqrt(x*x + y*y) ,

taking precautions against unwarranted IEEE exceptions. On IEEE overflow, **hypot( )** may also set **errno** and call **matherr**(3M). **hypot(±∞, y)** is +∞ for any y, even a NaN, and is exceptional only for a signaling NaN.

hypot(x,y) and **atan2**(y,x) (see **trig**(3M)) convert rectangular coordinates (x,y) to polar (r,θ); **hypot( )** computes r, the modulus or radius.

SEE ALSO

trig(3M), **matherr**(3M)

NAME

ieee_flags – mode and status function for IEEE standard arithmetic

SYNOPSIS

#include <sys/ieeefp.h>

int ieee_flags(action, mode, in, out)
char *action, *mode, *in, **out;

DESCRIPTION

This function provides easy access to the modes and status required to fully exploit ANSI/IEEE Std 754-1985 arithmetic in a C program. All arguments are pointers to strings. Results arising from invalid arguments and invalid combinations are undefined for efficiency.

There are four types of *action*: **get**, **set**, **clear** and **clearall**. There are three valid settings for *mode*, two corresponding to modes of IEEE arithmetic:

direction	current rounding direction mode
precision	current rounding precision mode

and one corresponding to status of IEEE arithmetic:

exception	accrued exception-occurred status

There are fourteen types of *in* and *out*:

nearest	round toward nearest
tozero	round toward zero
negative	round toward negative infinity
positive	round toward positive infinity
extended	
double	
single	
inexact	
division	division by zero exception
underflow	
overflow	
invalid	
all	all five exceptions above
common	invalid, overflow, and division exceptions

Note: **all** and **common** only make sense with **set** or **clear**.

For **clearall**, **ieee_flags( )** returns 0 and restores all default modes and status. Nothing will be assigned to *out*. Thus

```
char *mode, *out, *in;
ieee_flags("clearall", mode, in, &out);
```

set rounding direction to **nearest**, rounding precision to **extended**, and all accrued exception-occurred status to zero.

For **clear**, **ieee_flags( )** returns 0 and restores the default mode or status. Nothing will be assigned to *out*. Thus

char *out, *in;
ieee_flags("clear", "direction", in, &out);   . . . set rounding direction to round to nearest.

For **set**, **ieee_flags( )** returns 0 if the action is successful and 1 if the corresponding required status or mode is not available (for instance, not supported in hardware). Nothing will be assigned to *out*. Thus

char *out, *in;
ieee_flags ("set", "direction", "tozero", &out);   set rounding direction to round toward zero;

For **get**, we have the following cases:

Case 1: *mode* is **direction**. In that case, *out* returns one of the four strings **nearest**, **tozero**, **positive**, **negative**, and **ieee_flags( )** returns a value corresponding to *out* according to the **enum fp_direction_type** defined in **<sys/ieeefp.h>**.

Case 2: *mode* is **precision**. In that case, *out* returns one of the three strings **extended**, **double** and **single**, and **ieee_flags( )** returns a value corresponding to *out* according to the **enum fp_precision_type** defined in **<sys/ieeefp.h>**.

Case 3: *mode* is **exception**. In that case, *out* returns

not available   if information on exception is not available.

no exception   if no accrued exception.

the accrued exception that has the highest priority according to the following list:

the exception named by *in*
invalid
overflow
division
underflow
inexact

In this case **ieee_flags( )** returns a five or six bit value where each bit (see **enum fp_exception_type** in **<sys/ieeefp.h>**) corresponds to an exception-occurred accrued status flag: 0 = off, 1 = on. The bit corresponding to a particular exception varies among architectures (see **<sys/ieeefp.h>**).

Example:

char *out; int k, ieee_flags( );
ieee_flags("clear", "exception", "all", &out);   **/* clear all accrued exceptions */**
. . .
code that generates three exceptions: **overflow**, **invalid**, **inexact**
. . .
k = ieee_flags("get", "exception", "overflow", &out);

then *out* is **overflow**, and on a Sun-3, *k* is 25.

NAME

ieee_functions, fp_class, finite, ilogb, isinf, isnan, isnormal, issubnormal, iszero, signbit, copysign, fabs, fmod, nextafter, remainder, scalbn – appendix and related miscellaneous functions for IEEE arithmetic

SYNOPSIS

```
#include <math.h>
#include <stdio.h>

enum fp_class_type fp_class(x)
double x;

int finite(x)
double x;

int ilogb(x)
double x;

int isinf(x)
double x;

int isnan(x)
double x;

int isnormal(x)
double x;

int issubnormal(x)
double x;

int iszero(x)
double x;

int signbit(x)
double x;

void ieee_retrospective(f)
FILE *f;

void nonstandard_arithmetic()

void standard_arithmetic()

double copysign(x,y)
double x, y;

double fabs(x)
double x;

double fmod(x,y)
double x, y;

double nextafter(x,y)
double x, y;

double remainder(x,y)
double x, y;

double scalbn(x,n)
double x; int n;
```

DESCRIPTION

Most of these functions provide capabilities required by ANSI/IEEE Std 754-1985 or suggested in its appendix.

fp_class(x) corresponds to the IEEE's **class( ) and** classifies x as zero, subnormal, normal, ∞, or quiet or signaling *NaN*. **<floatingpoint.h>** defines **enum fp_class_type**. The following functions return 0 if the indicated condition is not satisfied:

finite(x)	returns 1 if x is zero, subnormal or normal
isinf(x)	returns 1 if x is ∞
isnan(x)	returns 1 if x is *NaN*
isnormal(x)	returns 1 if x is normal
issubnormal(x)	returns 1 if x is subnormal
iszero(x)	returns 1 if x is zero
signbit(x)	returns 1 if x's sign bit is set

ilogb(x) returns the unbiased exponent of x in integer format. **ilogb**($\pm\infty$) = +**MAXINT** and **ilogb(0)** = −**MAXINT**; **<values.h>** defines **MAXINT** as the largest int. **ilogb**(x) never generates an exception. When x is subnormal, **ilogb**(x) returns an exponent computed as if x were first normalized.

ieee_retrospective(*f*) prints a message to the **FILE** *f* listing all IEEE accrued exception-occurred bits currently on, unless no such bits are on or the only one on is "inexact". It's intended to be used at the end of a program to indicate whether some IEEE floating-point exceptions occurred that might have affected the result.

standard_arithmetic() and **nonstandard_arithmetic**() are meaningful on systems that provide an alternative faster mode of floating-point arithmetic that does not conform to the default IEEE Standard. Nonstandard modes vary among implementations; nonstandard mode may, for instance, result in setting subnormal results to zero or in treating subnormal operands as zero, or both, or something else. **standard_arithmetic**() reverts to the default standard mode. On systems that provide only one mode, these functions have no effect.

copysign(x,y) returns x with y's sign bit.

fabs(x) returns the absolute value of x.

nextafter(x,y) returns the next machine representable number from x in the direction y.

remainder(x, y) and **fmod**(x, y) return a remainder of x with respect to y; that is, the result r is one of the numbers that differ from x by an integral multiple of y. Thus $(x - r)/y$ is an integral value, even though it might exceed **MAXINT** if it were explicitly computed as an int. Both functions return one of the two such r smallest in magnitude. **remainder**(x, y) is the operation specified in ANSI/IEEE Std 754-1985; the result of **fmod**(x, y) may differ from **remainder**( )'s result by $\pm y$. The magnitude of **remainder**'s result can not exceed half that of y; its sign might not agree with either x or y. The magnitude of **fmod**( )'s result is less than that of y; its sign agrees with that of x. Neither function can generate an exception as long as both arguments are normal or subnormal. **remainder**(x, **0**), **fmod**(x, **0**), **remainder**(∞, y), and **fmod**(∞, y) are invalid operations that produce a *NaN*.

scalbn(x, n) returns x* **2**n** computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Thus

$$1 \le \mathbf{scalbn}(\mathbf{fabs}(x), -\mathbf{ilogb}(x)) < 2$$

for every x except 0, ∞, and *NaN*.

SEE ALSO

floatingpoint(3), **ieee_flags**(3M), **matherr**(3M)

NAME

ieee_handler – IEEE exception trap handler function

SYNOPSIS

#include <floatingpoint.h>

int ieee_handler(action,exception,hdl)
char action[ ], exception[ ];
sigfpe_handler_type hdl;

DESCRIPTION

This function provides easy exception handling to exploit ANSI/IEEE Std 754-1985 arithmetic in a C program. The first two arguments are pointers to strings. Results arising from invalid arguments and invalid combinations are undefined for efficiency.

There are three types of *action* : **get, set,** and **clear**. There are five types of *exception* :

inexact	
division	... division by zero exception
underflow	
overflow	
invalid	
all	... all five exceptions above
common	... invalid, overflow, and division exceptions

Note: **all** and **common** only make sense with **set** or **clear**.

hdl contains the address of a signal-handling routine. **<floatingpoint.h>** defines *sigfpe_handler_type*.

get will return the location of the current handler routine for *exception* cast to an int. **set** will set the routine pointed at by **hdl** to be the handler routine and at the same time enable the trap on *exception*, except when **hdl** == **SIGFPE_DEFAULT** or **SIGFPE_IGNORE**; then **ieee_handler( )** will disable the trap on *exception*. When **hdl** == **SIGFPE_ABORT**, any trap on *exception* will dump core using **abort**(3). **clear all** disables trapping on all five exceptions.

Two steps are required to intercept an IEEE-related **SIGFPE** code with **ieee_handler**:

1) Set up a handler with **ieee_handler**.

2) Perform a floating-point operation that generates the intended IEEE exception.

Unlike **sigfpe**(3), **ieee_handler( )** also adjusts floating-point hardware mode bits affecting IEEE trapping. For **clear**, **set SIGFPE_DEFAULT**, or **set SIGFPE_IGNORE**, the hardware trap is disabled. For any other **set**, the hardware trap is enabled.

SIGFPE signals can be handled using **sigvec**(2), **signal**(3V), **sigfpe**(3), or **ieee_handler**(3M). In a particular program, to avoid confusion, use only one of these interfaces to handle **SIGFPE** signals.

DIAGNOSTICS

ieee_handler( ) normally returns 0 for **set**. 1 will be returned if the action is not available (for instance, not supported in hardware). For **get**, the address of the current handler is returned, cast to an int.

EXAMPLE

A user-specified signal handler might look like this:

```
void sample_handler(sig, code, scp, addr)
int sig;          /* sig == SIGFPE always */
int code;
struct sigcontext *scp;
char *addr;
{
        /*
         * Sample user-written sigfpe code handler.
         * Prints a message and continues.
         * struct sigcontext is defined in <signal.h>.
         */
        printf("ieee exception code %x occurred at pc %X \n", code, scp->sc_pc);
}
```

and it might be set up like this:

```
extern void sample_handler();
main()
{
        sigfpe_handler_type hdl, old_handler1, old_handler2;
        /*
         * save current overflow and invalid handlers
         */
        old_handler1 = (sigfpe_handler_type) ieee_handler("get", "overflow", old_handler1);
        old_handler2 = (sigfpe_handler_type) ieee_handler("get", "invalid", old_handler2);
        /*
         * set new overflow handler to sample_handler() and set new
         * invalid handler to SIGFPE_ABORT (abort on invalid)
         */
        hdl = (sigfpe_handler_type) sample_handler;
        if (ieee_handler("set", "overflow", hdl) != 0)
                printf("ieee_handler can't set overflow \n");
        if (ieee_handler("set", "invalid", SIGFPE_ABORT) != 0)
                printf("ieee_handler can't set invalid \n");
        ...
        /*
         * restore old overflow and invalid handlers
         */
        ieee_handler("set", "overflow", old_handler1);
        ieee_handler("set", "invalid", old_handler2);
}
```

SEE ALSO

sigvec(2), **abort**(3), **floatingpoint**(3), **sigfpe**(3), **signal**(3V)

NAME

ieee_test, logb, scalb, significant – IEEE test functions for verifying standard compliance

SYNOPSIS

#include <math.h>

double logb(x)
double x;

double scalb(x,y)
double x; double y;

double significant(x)
double x;

DESCRIPTION

These functions allow users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California. Their use is not otherwise recommended; instead use **scalbn**(*x*,*n*) and **ilogb**(*x*) described in **ieee_functions**(3M). See the *Numerical Computation Guide* for details.

logb(*x*) returns the unbiased exponent of *x* in floating-point format, for exercising the logb(L) test vector. **logb**(±∞) = +∞; **logb(0)** = –∞ with a division by zero exception. **logb**(*x*) differs from **ilogb**(*x*) in returning a result in floating-point rather than integer format, in sometimes signaling IEEE exceptions, and in not normalizing subnormal *x*.

scalb(*x***,(double)n***)* returns *x* * *2**n* computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication, for exercising the scalb(S) test vector. Thus

0 ≤ scalb(fabs(*x***),–logb(***x***)) < 2**

for every *x* except 0, ∞ and *NaN*. **scalb**(*x*,*y*) is not defined when *y* is not an integral value. **scalb**(*x*,*y*) differs from **scalbn(***x***,n***)* in that the second argument is in floating-point rather than integer format.

significant(*x*) computes just

scalb(*x***, (double) -ilogb(***x***)),**

for exercising the fraction-part(F) test vector.

FILES

/usr/lib/libm.a

SEE ALSO

floatingpoint(3), **ieee_values**(3M), **ieee_functions**(3M), **matherr**(3M)

NAME

ieee_values, min_subnormal, max_subnormal, min_normal, max_normal, infinity, quiet_nan, signaling_nan, HUGE, HUGE_VAL – functions that return extreme values of IEEE arithmetic

SYNOPSIS

#include <math.h>

double min_subnormal()

double max_subnormal()

double min_normal()

double max_normal()

double infinity()

double quiet_nan(n)
long n;

double signaling_nan(n)
long n;

#define HUGE (infinity())

#define HUGE_VAL (infinity())

DESCRIPTION

These functions return special values associated with ANSI/IEEE Std 754-1985 double-precision floating-point arithmetic: the smallest and largest positive subnormal numbers, the smallest and largest positive normalized numbers, positive infinity, and a quiet and signaling NaN. The long parameters *n* to **quiet_nan**(*n*) and **signaling_nan**(*n*) are presently unused but are reserved for future use to specify the significant of the returned NaN.

None of these functions are affected by IEEE rounding or trapping modes or generate any IEEE exceptions.

The macro **HUGE** returns $+\infty$ in accordance with previous SunOS releases. The macro **HUGE_VAL** returns $+\infty$ in accordance with the System V Interface Definition.

FILES

/usr/lib/libm.a

SEE ALSO

ieee_functions(3M)

NAME

lgamma – log gamma function

SYNOPSIS

#include <math.h>

extern int signgam;

double lgamma(x)
double x;

DESCRIPTION

lgamma( ) returns

$$\ln |\Gamma(x)|$$

where

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

for x > 0 and

$$\Gamma(x) = \pi/(\Gamma(1-x)\sin(\pi x))$$

for x < 1.

The external integer **signgam** returns the sign of $\Gamma(x)$.

IDIOSYNCRASIES

Do *not* use the expression **signgam*exp(lgamma(x))** to compute '**g** := Γ(**x**)'. Instead compute **lgamma( )** first:

lg = lgamma(x); g = signgam*exp(lg);

only after **lgamma( )** has returned can **signgam** be correct. Note: $\Gamma(x)$ must overflow when x is large enough, underflow when $-x$ is large enough, and generate a division by zero exception at the singularities x a nonpositive integer. In addition, **lgamma( )** may also set **errno** and call **matherr**(3M).

SEE ALSO

matherr(3M)

NAME

matherr – math library exception-handling function

SYNOPSIS

#include <math.h>

int matherr(exc)
struct exception *exc;

DESCRIPTION

The SVID (*System V Interface Definition*) specifies that certain **libm** functions call **matherr( )** when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named **matherr( )** in their programs. **matherr( )** is of the form described above. When an exception occurs, a pointer to the exception structure *exc* will be passed to the user-supplied **matherr( )** function. This structure, which is defined in the **<math.h>** header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element **type** is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):

DOMAIN	argument domain exception
SING	argument singularity
OVERFLOW	overflow range exception
UNDERFLOW	underflow range exception

The element **name** points to a string containing the name of the function that incurred the exception. The elements **arg1** and **arg2** are the arguments with which the function was invoked. **retval** is set to the default value that will be returned by the function unless the user's **matherr( )** sets it to a different value.

If the user's **matherr( )** function returns non-zero, no exception message will be printed, and **errno** will not be set.

If **matherr( )** is not supplied by the user, the default matherr exception-handling mechanisms, summarized in the table below, will be invoked upon exception:

DOMAIN==fp_invalid

An IEEE NaN is usually returned, **errno** is set to EDOM, and a message is printed on standard error. **pow(***0.0***,0.0)** and **atan2(0.0,0.0)** return numerical default results but set **errno** and print the message.

SING==fp_division

An IEEE ∞ of appropriate sign is returned, **errno** is set to EDOM, and a message is printed on standard error.

OVERFLOW==fp_overflow

In the default rounding direction, an IEEE ∞ of appropriate sign is returned. In optional rounding directions, ±**MAXDOUBLE**, the largest finite double-precision number, is sometimes returned instead of ±∞. **errno** is set to ERANGE.

UNDERFLOW==fp_underflow

An appropriately-signed zero, subnormal number, or smallest normalized number is returned, and **errno** is set to ERANGE.

The facilities provided by **matherr( )** are not available in situations such as compiling on a Sun-3 system with **/usr/lib/f68881/libm.il** or **/usr/lib/ffpa/libm.il**, in which case some **libm** functions are converted to atomic hardware operations. In these cases setting **errno** and calling **matherr( )** are not worth the adverse performance impact, but regular ANSI/IEEE Std 754-1985 exception handling remains available. In any case **errno** is not a reliable error indicator in that it may be unexpectedly set by a function in a handler for an asynchronous signal.

DEFAULT ERROR HANDLING PROCEDURES				
	Types of Errors			
<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW
errno	EDOM	EDOM	ERANGE	ERANGE
IEEE Exception	Invalid Operation	Division by Zero	Overflow	Underflow
<floatingpoint.h> type	fp_invalid	fp_division	fp_overflow	fp_underflow
ACOS, ASIN:	M, NaN	–	–	–
ATAN2(0,0):	M, ±0.0 or ±π	–	–	–
BESSEL:				
y0, y1, yn (x < 0)	M, NaN	–	–	–
y0, y1, yn (x = 0)	–	M, −∞	–	–
COSH, SINH:	–	–	IEEE Overflow	–
EXP:	–	–	IEEE Overflow	IEEE Underflow
HYPOT:	–	–	IEEE Overflow	–
LGAMMA:	–	M, +∞	IEEE Overflow	–
LOG, LOG10:				
(x < 0)	M, NaN	–	–	–
(x = 0)	–	M, −∞	–	–
POW:				
usual cases	–	–	IEEE Overflow	IEEE Underflow
(x < 0) ** (y not an integer)	M, NaN	–	–	–
0 ** 0	M, 1.0	–	–	–
0 ** (y < 0)	–	M, ±∞	–	–
SQRT:	M, NaN	–	–	–

ABBREVIATIONS	
M	Message is printed (EDOM exception).
NaN	IEEE NaN result and invalid operation exception.
∞	IEEE ∞ result and division-by-zero exception.
IEEE Overflow	IEEE Overflow result and exception.
IEEE Underflow	IEEE Underflow result and exception.
π	Closest machine-representable approximation to **pi**.

The interaction of IEEE arithmetic and **matherr**( ) is not defined when executing under IEEE rounding modes other than the default round to nearest: **matherr**( ) may not be called on overflow or underflow, and the Sun-provided **matherr**( ) may return results that differ from those in this table.

EXAMPLE

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
        switch (x->type) {
        case
                DOMAIN:
                /* change sqrt to return sqrt(-arg1), not NaN */
                if (!strcmp(x->name, "sqrt")) {
                        x->retval = sqrt(-x->arg1);
                        return (0); /* print message and set errno */
        } /* fall through */
        case
                SING:
                /* all other domain or sing exceptions, print message and abort */
                fprintf(stderr, "domain exception in %s\n", x->name);
                abort( );
                break;
        }
        return (0); /* all other exceptions, execute default procedure */
}
```

NAME

aint, anint, ceil, floor, rint, irint, nint – round to integral value in floating-point or integer format

SYNOPSIS

```
#include <math.h>

double aint(x)
double x;

double anint(x)
double x;

double ceil(x)
double x;

double floor(x)
double x;

double rint(x)
double x;

int irint(x)
double x;

int nint(x)
double x;
```

DESCRIPTION

aint(), **anint()**, **ceil()**, **floor()**, and **rint()** convert a double value into an integral value in double format. They vary in how they choose the result when the argument is not already an integral value. Here an "integral value" means a value of a mathematical integer, which however might be too large to fit in a particular computer's int format. All sufficiently large values in a particular floating-point format are already integral; in IEEE double-precision format, that means all values >= 2**52. Zeros, infinities, and quiet NaNs are treated as integral values by these functions, which always preserve their argument's sign.

aint() returns the integral value between *x* and 0, nearest *x*. This corresponds to IEEE rounding toward zero and to the Fortran generic intrinsic function **aint()**.

anint() returns the nearest integral value to *x*, except halfway cases are rounded to the integral value larger in magnitude. This corresponds to the Fortran generic intrinsic function **anint()**.

ceil() returns the least integral value greater than or equal to *x*. This corresponds to IEEE rounding toward positive infinity.

floor() returns the greatest integral value less than or equal to *x*. This corresponds to IEEE rounding toward negative infinity.

rint() rounds *x* to an integral value according to the current IEEE rounding direction.

irint() converts *x* into int format according to the current IEEE rounding direction.

nint() converts *x* into int format rounding to the nearest int value, except halfway cases are rounded to the int value larger in magnitude. This corresponds to the Fortran generic intrinsic function **nint()**.

NAME

single_precision – single-precision access to libm functions

SYNOPSIS

#include <math.h>

FLOATFUNCTIONTYPE r_acos_ (x)
FLOATFUNCTIONTYPE r_acospi_ (x)
FLOATFUNCTIONTYPE r_acosh_ (x)
FLOATFUNCTIONTYPE r_aint_ (x)
FLOATFUNCTIONTYPE r_anint_ (x)
FLOATFUNCTIONTYPE r_annuity_ (x)
FLOATFUNCTIONTYPE r_asin_ (x)
FLOATFUNCTIONTYPE r_asinpi_ (x)
FLOATFUNCTIONTYPE r_asinh_ (x)
FLOATFUNCTIONTYPE r_atan_ (x)
FLOATFUNCTIONTYPE r_atanpi_ (x)
FLOATFUNCTIONTYPE r_atanh_ (x)
FLOATFUNCTIONTYPE r_atan2_ (x,y)
FLOATFUNCTIONTYPE r_atan2pi_ (x,y)
FLOATFUNCTIONTYPE r_cbrt_ (x)
FLOATFUNCTIONTYPE r_ceil_ (x)
enum fp_class_type ir_fp_class_ (x)
FLOATFUNCTIONTYPE r_compound_ (x,y)
FLOATFUNCTIONTYPE r_copysign_ (x,y)
FLOATFUNCTIONTYPE r_cos_ (x)
FLOATFUNCTIONTYPE r_cospi_ (x)
FLOATFUNCTIONTYPE r_cosh_ (x)
FLOATFUNCTIONTYPE r_erf_ (x)
FLOATFUNCTIONTYPE r_erfc_ (x)
FLOATFUNCTIONTYPE r_exp_ (x)
FLOATFUNCTIONTYPE r_expm1_ (x)
FLOATFUNCTIONTYPE r_exp2_ (x)
FLOATFUNCTIONTYPE r_exp10_ (x)
FLOATFUNCTIONTYPE r_fabs_ (x)
int ir_finite_ (x)
FLOATFUNCTIONTYPE r_floor_ (x)
FLOATFUNCTIONTYPE r_fmod_ (x,y)
FLOATFUNCTIONTYPE r_hypot_ (x,y)
int ir_ilogb_ (x)
int ir_irint_ (x)
int ir_isinf_ (x)
int ir_isnan_ (x)
int ir_isnormal_ (x)
int ir_issubnormal_ (x)
int ir_iszero_ (x)
int ir_nint_ (x)
FLOATFUNCTIONTYPE r_infinity_ ( )
FLOATFUNCTIONTYPE r_j0_ (x)
FLOATFUNCTIONTYPE r_j1_ (x)
FLOATFUNCTIONTYPE r_jn_ (n,x)
FLOATFUNCTIONTYPE r_lgamma_ (x)
FLOATFUNCTIONTYPE r_logb_ (x)
FLOATFUNCTIONTYPE r_log_ (x)
FLOATFUNCTIONTYPE r_log1p_ (x)

FLOATFUNCTIONTYPE r_log2_ (x)
FLOATFUNCTIONTYPE r_log10_ (x)
FLOATFUNCTIONTYPE r_max_normal_ ()
FLOATFUNCTIONTYPE r_max_subnormal_ ()
FLOATFUNCTIONTYPE r_min_normal_ ()
FLOATFUNCTIONTYPE r_min_subnormal_ ()
FLOATFUNCTIONTYPE r_nextafter_ (x,y)
FLOATFUNCTIONTYPE r_pow_ (x,y)
FLOATFUNCTIONTYPE r_quiet_nan_ (n)
FLOATFUNCTIONTYPE r_remainder_ (x,y)
FLOATFUNCTIONTYPE r_rint_ (x)
FLOATFUNCTIONTYPE r_scalb_ (x,y)
FLOATFUNCTIONTYPE r_scalbn_ (x,n)
FLOATFUNCTIONTYPE r_signaling_nan_ (n)
int ir_signbit_ (x)
FLOATFUNCTIONTYPE r_significant_ (x)
FLOATFUNCTIONTYPE r_sin_ (x)
FLOATFUNCTIONTYPE r_sinpi_ (x)
void r_sincos_ (x,s,c)
void r_sincospi_ (x,s,c)
FLOATFUNCTIONTYPE r_sinh_ (x)
FLOATFUNCTIONTYPE r_sqrt_ (x)
FLOATFUNCTIONTYPE r_tan_ (x)
FLOATFUNCTIONTYPE r_tanpi_ (x)
FLOATFUNCTIONTYPE r_tanh_ (x)
FLOATFUNCTIONTYPE r_y0_ (x)
FLOATFUNCTIONTYPE r_y1_ (x)
FLOATFUNCTIONTYPE r_yn_ (n,x)

float *x, *y, *s, *c
int *n

DESCRIPTION

These functions are single-precision versions of certain **libm** functions. Primarily for use by Fortran programmers, these functions may also be used in other languages. The single-precision floating-point results are deviously declared to avoid C's automatic type conversion to double.

FILES

/usr/lib/libm.a

NAME

sqrt, cbrt – cube root, square root

SYNOPSIS

#include <math.h>

double cbrt(x)
double x;

double sqrt(x)
double x;

DESCRIPTION

sqrt(*x*) returns the square root of *x*, correctly rounded according to ANSI/IEEE 754-1985. In addition, **sqrt**( ) may also set **errno** and call **matherr**(3M).

cbrt(*x*) returns the cube root of *x*. **cbrt**( ) is accurate to within 0.7 *ulp*s.

SEE ALSO

matherr(3M)

NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

void sincos(x, s, c)
double x, *s, *c;

double tan(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(y, x)
double y, x;

double sinpi(x)
double x;

double cospi(x)
double x;

void sincospi(x, s, c)
double x, *s, *c;

double tanpi(x)
double x;

double asinpi(x)
double x;

double acospi(x)
double x;

double atanpi(x)
double x;

double atan2pi(y, x)
double y, x;

DESCRIPTION

sin( ), **cos**( ), **sincos**( ), and **tan**( ) return trigonometric functions of radian arguments. The values of trigonometric functions of arguments exceeding $\pi/4$ in magnitude are affected by the precision of the approximation to $\pi/2$ used to reduce those arguments to the range $-\pi/4$ to $\pi/4$. Argument reduction may occur in hardware or software; if in software, the variable **fp_pi** defined in **<math.h>** allows changing that precision at run time. Trigonometric argument reduction is discussed in the *Numerical Computation Guide*. Note: **sincos(x,s,c)** allows simultaneous computation of ***s** = **sin(x)** and ***c** = **cos(x)**.

asin( ) returns the arc sin in the range $-\pi/2$ to $\pi/2$.

acos( ) returns the arc cosine in the range 0 to π.

atan( ) returns the arc tangent of *x* in the range $-\pi/2$ to $\pi/2$.

atan2(*y*,*x***)** and **hypot(x,y)** (see **hypot**(3M)) convert rectangular coordinates (x,y) to polar (r,θ); **atan2( )** computes θ, the argument or phase, by computing an arc tangent of y/x in the range $-\pi$ to π. **atan2**(0.0,0.0) is ±0.0 or $\pm\pi$, in conformance with 4.3BSD, as discussed in the *Numerical Computation Guide*.

sinpi( ), **cospi( )**, and **tanpi( )** avoid range-reduction issues because their definition **sinpi(x)==sin(**π*** x)** permits range reduction that is fast and exact for all *x*. The corresponding inverse functions compute **asinpi(x)==asin(x)/**π. Similarly **atan2pi(y,x)==atan2(y,x)/**π.

DIAGNOSTICS

These functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. **sin(**$\pm\infty$**)**, **cos(**$\pm\infty$**)**, **tan(**$\pm\infty$**)**, or **asin(***x***)** or **acos(***x***)** with $|x|>1$, return NaN; **sinpi(***x***)** et. al. are similar. In addition, **asin( )**, **acos( )**, and **atan2( )** may also set **errno** and call **matherr**(3M).

SEE ALSO

hypot(3M), **matherr**(3M)

NAME

intro – introduction to RPC service library functions and protocols

DESCRIPTION

These functions constitute the RPC service library. Most of these describe RPC protocols. The PROTOCOL section describes how to access the protocol description file. This file may be compiled with **rpcgen**(1) to produce data definitions and XDR routines. Procompiled versions of header files sometimes exist as **<rpcsvc/*.h>** and precompiled XDR routines and programming interfaces to the protocols sometimes exist in *librpcsvc*. Warning: some of these header files and XDR routines were hand-written because they existed before *rpcgen*. They do not correspond to their protocol description file. In order to get the link editor to load this library, use the **–lrpcsvc** option of **cc**(1V). Information about the availability of programming interfaces to these protocols is available under PROGRAMMING section of each manual page.

Some routines in the **librpcsvc** library do not correspond to protocols, but are useful utilities for RPC programming. These are distinguished by the presence of the SYNOPSIS section instead of the usual PROTOCOL section.

LIST OF STANDARD RPC SERVICES

Name	Appears on Page	Description
bootparam	**bootparam**(3R)	bootparam protocol
ether	**ether**(3R)	monitor traffic on the Ethernet
getpublickey	**publickey**(3R)	get public or secret key
getrpcport	**getrpcport**(3R)	get RPC port number
getsecretkey	**publickey**(3R)	get public or secret key
ipalloc	**ipalloc**(3R)	determine or temporarily allocate IP address
klm_prot	**klm_prot**(3R)	protocol between kernel and local lock manager
mount	**mount**(3R)	keep track of remotely mounted filesystems
nlm_prot	**nlm_prot**(3R)	protocol between local and remote network lock managers
passwd2des	**xcrypt**(3R)	hex encryption and utility routines
pnp	**pnp**(3R)	automatic network installation
publickey	**publickey**(3R)	get public or secret key
rex	**rex**(3R)	remote execution protocol
rnusers	**rnusers**(3R)	return information about users on remote machines
rquota	**rquota**(3R)	implement quotas on remote machines
rstat	**rstat**(3R)	get performance data from remote kernel
rusers	**rnusers**(3R)	return information about users on remote machines
rwall	**rwall**(3R)	write to specified remote machines
sm_inter	**sm_inter**(3R)	status monitor protocol
spray	**spray**(3R)	scatter data in order to check the network
xcrypt	**xcrypt**(3R)	hex encryption and utility routines
xdecrypt	**xcrypt**(3R)	hex encryption and utility routines
xencrypt	**xcrypt**(3R)	hex encryption and utility routines
yp	**yp**(3R)	NIS protocol
yppasswd	**yppasswd**(3R)	update user password in NIS

NAME
bootparam – bootparam protocol

PROTOCOL
/usr/include/rpcsvc/bootparam_prot.x

DESCRIPTION
The bootparam protocol is used for providing information to the diskless clients necessary for booting.

PROGRAMMING
#include <rpcsvc/bootparam.h>

XDR Routines
The following XDR routines are available in **librpcsvc**:

xdr_bp_whoami_arg
xdr_bp_whoami_res
xdr_bp_getfile_arg
xdr_bp_getfile_res

SEE ALSO
bootparams(5), **bootparamd**(8)

NAME

ether – monitor traffic on the Ethernet

PROTOCOL

/usr/include/rpcsvc/ether.x

DESCRIPTION

The ether protocol is used for monitoring traffic on the ethernet.

PROGRAMMING

#include <rpcsvc/ether.h>

The following XDR routines are available in **librpcsvc**:

xdr_etherstat
xdr_etheraddrs
xdr_etherhtable
xdr_etherhmem
xdr_addrmask

SEE ALSO

traffic(1C), **etherfind**(8C), **etherd**(8C)

NAME

getrpcport – get RPC port number

SYNOPSIS

int getrpcport(host, prognum, versnum, proto)
char *host;
int prognum, versnum, proto;

DESCRIPTION

getrpcport() returns the port number for version *versnum* of the RPC program *prognum* running on *host* and using protocol *proto*. It returns 0 if it cannot contact the portmapper, or if *prognum* is not registered. If *prognum* is registered but not with version *versnum*, it will still return a port number (for some version of the program) indicating that the program is indeed registered. The version mismatch will be detected upon the first call to the service.

NAME

ipalloc – determine or temporarily allocate IP address

PROTOCOL

/usr/include/rpcsvc/ipalloc.x

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ipalloc( ) is the protocol for allocating the IP address that a system should use.

PROGRAMMING

#include <rpcsvc/ipalloc.h>

The following RPC calls are available in version 2 of this protocol:

NULLPROC

This is a standard null entry, used to ping a service to measure overhead or to discover servers.

IP_ALLOC

Returns an IP address corresponding to a given Ethernet address, if possible. This RPC must be called using DES authentication, from a client authorized to allocate IP addresses. A cache of allocated addresses is maintained.

The first action taken on receipt of this RPC is to verify that no existing mapping between the *etheraddr* and the *netnum* exists in the Network Information Service (NIS) database. If one is found, then that is returned. Otherwise, an internal cache is checked, and if an entry is found there for the given *etheraddr* on the right network, that entry is used. If no address was found either in the NIS database or in the cache, a new one may be allocated and returned, and the *ip_success* status is returned.

If an unusable entry was found in the cache, this RPC returns **ip_failure** status.

IP_TONAME

Used to determine whether a given IP address is known to the NIS service, since NIS allows a delay between the posting of an address and its availability in some locations on the network.

IP_FREE

This RPC is used to delete *ipaddr* entries from the cache when they are no longer needed there. It requires the same protections as the **IP_ALLOC** RPC.

SEE ALSO

ipallocd(8C), **pnpboot**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

klm_prot – protocol between kernel and local lock manager

PROTOCOL

/usr/include/klm_prot.x

DESCRIPTION

The protocol is used for communication between kernel and local lock manager.

PROGRAMMING

#include <rpcsvc/klm_prot.h>

XDR Routines

The following XDR routines are available in **librpcsvc**:

xdr_klm_testargs
xdr_klm_testrply
xdr_klm_lockargs
xdr_klm_unlockargs
xdr_klm_stat

SEE ALSO

lockd(8C)

NAME

mount – keep track of remotely mounted filesystems

PROTOCOL

/usr/include/rpcsvc/mount.x

DESCRIPTION

The mount protocol is separate from, but related to, the NFS protocol. It provides all of the operating system specific services to get the NFS off the ground — looking up path names, validating user identity, and checking access permissions. Clients use the mount protocol to get the first file handle, which allows them entry into a remote filesystem.

The mount protocol is kept separate from the NFS protocol to make it easy to plug in new access checking and validation methods without changing the NFS server protocol.

Note: the protocol definition implies stateful servers because the server maintains a list of client's mount requests. The mount list information is not critical for the correct functioning of either the client or the server. It is intended for advisory use only, for example, to warn people when a server is going down.

PROGRAMMING

#include <rpcsvc/mount.h>

The following XDR routines are available in **librpcsvc**:
xdr_exportbody
xdr_exports
xdr_fhandle
xdr_fhstatus
xdr_groups
xdr_mountbody
xdr_mountlist
xdr_path

SEE ALSO

mount(8), **mountd**(8C), **showmount**(8)

NFS Protocol Spec, in *Network Programming*

NAME

nlm_prot – protocol between local and remote network lock managers

PROTOCOL

/usr/include/rpcsvc/nlm_prot.x

DESCRIPTION

The network lock manager protocol is used for communication between local and remote lock managers.

PROGRAMMING

#include <rpcsvc/nlm_prot.h>

XDR Routines

The following XDR routines are available in **librpcsvc**:

xdr_nlm_testargs
xdr_nlm_testres
xdr_nlm_lockargs
xdr_nlm_cancargs
xdr_nlm_unlockargs
xdr_nlm_res

SEE ALSO

lockd(8C)

NAME

pnp – automatic network installation

PROTOCOL

/usr/include/rpcsvc/pnprpc.x

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

pnp() is used during unattended network installation, and routine booting, of Sun386i systems on a Sun386i network. Each network cable (subnetwork or full network) must have at least one **pnpd**(8C) server running on it to support PNP.

PROGRAMMING

#include <rpcsvc/pnprpc.h>

The following RPC calls are available in version 2 of the PNP protocol:

NULLPROC
: Finds a PNP daemon on the local network. Used with **clntudp_broadcast()**, often to measure network overhead.

PNP_WHOAMI
: Used early in the boot process to acquire network configuration information about a system, or to determine that a system is not known by the network.

PNP_ACQUIRE
: Used to acquire a server willing to configure a new system after a **PNP_WHOAMI** request fails. This RPC is typically broadcast; any successful reply may be used.

PNP_SETUP
: Requests a network configuration from a PNP daemon that has responded to a previous **PNP_ACQUIRE** RPC.

PNP_POLL
: After a **PNP_SETUP** request, if the status is **in_progress**, the procedure is to wait 20 seconds, and issue a **PNP_POLL** request, and then check the status again. Once the status is **success**, the system will be configured for the network. Entries in the yp database may be added or old ones deleted, and file storage may be assigned, according to the architecture and boot type.

If the server misses 5 **PNP_POLL** requests, it will assume that the client system crashed and back out of the procedure. Similarly, if the client system does not receive responses from the server for **PNP_MISSEDPOLLS** consecutive requests, it should assume the server crashed and begin its PNP sequence again.

SEE ALSO

pnpboot(8C), **pnpd**(8C)

NAME

publickey, getpublickey, getsecretkey – get public or secret key

SYNOPSIS

```
#include <rpc/rpc.h>
#include <rpc/key_prot.h>

getpublickey(netname, publickey)
char netname[MAXNETNAMELEN+1];
char publickey[HEXKEYBYTES+1];

getsecretkey(netname, secretkey, passwd)
char netname[MAXNETNAMELEN+1];
char secretkey[HEXKEYBYTES+1];
char *passwd;
```

DESCRIPTION

These routines are used to get public and secret keys from the YP database. **getsecretkey()** has an extra argument, *passwd*, which is used to decrypt the encrypted secret key stored in the database. Both routines return 1 if they are successful in finding the key, 0 otherwise. The keys are returned as NULL-terminated, hexadecimal strings. If the password supplied to **getsecretkey()** fails to decrypt the secret key, the routine will return 1 but the *secretkey* argument will be a NULL string.

SEE ALSO

publickey(5)

RPC Programmer's Manual in *Network Programming*

NAME

rex – remote execution protocol

PROTOCOL

/usr/include/rpcsvc/rex.x

DESCRIPTION

This server will execute commands remotely. The working directory and environment of the command can be specified, and the standard input and output of the command can be arbitrarily redirected. An option is provided for interactive I/O for programs that expect to be running on terminals. Note: this service is only provided with the TCP transport.

PROGRAMMING

```
#include <sys/ioctl.h>
#include <rpcsvc/rex.h>  /* not compiled with rpgen */
```

The following XDR routines are available in **librpcsvc**:

xdr_rex_start( )
xdr_rex_result( )
xdr_rex_ttymode( )
xdr_rex_ttysize( )

SEE ALSO

on(1C), **rexd**(8C)

NAME
rnusers, rusers – return information about users on remote machines

PROTOCOL
/usr/include/rpcsvc/rnusers.x

DESCRIPTION
rnusers( ) returns the number of users logged on to *host* (–1 if it cannot determine that number). **rusers( )** fills the **utmpidlearr** structure with data about *host*, and returns 0 if successful.

PROGRAMMING
```
#include <rpcsvc/rusers.h>
rnusers(host)
char *host
rusers(host, up)
char *host
struct utmpidlearr *up;
```

The following XDR routines are also available:
xdr_utmpidle
xdr_utmpidlearr

SEE ALSO
rusers(1C)

NAME

rquota – implement quotas on remote machines

PROTOCOL

/usr/include/rpcsvc/rquota.x

DESCRIPTION

The **rquota( )** protocol inquires about quotas on remote machines. It is used in conjunction with NFS, since NFS itself does not implement quotas.

PROGRAMMING

#include <rpcsvc/rquota.h>

The following XDR routines are available in **librpcsvc**:
xdr_getquota_arg
xdr_getquota_rslt
xdr_rquota

SEE ALSO

quota(1), **quotactl**(2)

NAME

rstat – get performance data from remote kernel

PROTOCOL

/usr/include/rpcsvc/rstat.x

DESCRIPTION

The **rstat()** protocol is used to gather statistics from remote kernel. Statistics are available on items such as paging, swapping and cpu utilization.

PROGRAMMING

#include <rpcsvc/rstat.h>

havedisk(host)
char *host;

rstat(host, statp)
char *host;
struct statstime *statp;

havedisk() returns 1 if *host* has a disk, 0 if it does not, and –1 if this cannot be determined. **rstat()** fills in the **statstime** structure for *host*, and returns 0 if it was successful.

The following XDR routines are available in **librpcsvc**:
xdr_statstime
xdr_statsswtch
xdr_stats

SEE ALSO

perfmeter(1), **rup**(1C), **rstatd**(8C)

NAME

rwall – write to specified remote machines

SYNOPSIS

#include <rpcsvc/rwall.h>

rwall(host, msg);
char *host, *msg;

DESCRIPTION

host prints the string *msg* to all its users. It returns 0 if successful.

RPC INFO

program number:
WALLPROG

procs:
WALLPROC_WALL
Takes string as argument (wrapstring), returns no arguments.
Executes *wall* on remote host with string.

versions:
RSTATVERS_ORIG

SEE ALSO

rwall(1C), **rwalld**(8C), **shutdown**(8)

NAME
PROTOC
DESCRII
PROGRA
XDR
SEE ALS

NAME

yppasswd – update user password in NIS

PROTOCOL

/usr/include/rpcsvc/yppasswd.x

DESCRIPTION

The **yppasswd()** protocol is used to change a user's password entry in the Network Inforn (NIS) password database.

If *oldpass* is indeed the old user password, this routine replaces the password entry with *new* 0 if successful.

PROGRAMMING

```
#include <rpcsvc/yppasswd.h>

yppasswd(oldpass, newpw)
        char *oldpass
        struct passwd *newpw;
```

SEE ALSO

yppasswd(1), **yppasswdd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). Th of the two remains the same; only the name has changed. The name Yellow Pages is a re mark in the United Kingdom of British Telecommunications plc, and may not be used witho

NAME

spray – scatter data in order to check the network

PROTOCOL

/usr/include/rpcsvc/spray.x

DESCRIPTION

The spray protocol sends packets to a given machine to test the speed and reliability of it.

PROGRAMMING

#include <rpcsvc/spray.h>

The following XDR routines are available in **librpcsvc**:

xdr_sprayarr
xdr_spraycumul

SEE ALSO

spray(8C), **sprayd**(8C)

NAME

xcrypt, xencrypt, xdecrypt, passwd2des – hex encryption and utility routines

SYNOPSIS

```
xencrypt(data, key)
char *data;
char *key;

xdecrypt(data, key)
char *data;
char *key;

passwd2des(pass, key)
char *pass;
char *key;
```

DESCRIPTION

The routines **xencrypt** and **xdecrypt** take null-terminated hexadecimal strings as arguments, and encrypt them using the 8-byte *key* as input to the DES algorithm. The input strings must have a length that is a multiple on 16 hex digits (64 bits is the DES block size).

passwd2des converts a password, of arbitrary length, into an 8-byte DES key, with odd-parity set in the low bit of each byte. The high-order bit of each input byte is ignored.

These routines are used by the DES authentication subsystem for encrypting and decrypting the secret keys stored in the publickey database.

SEE ALSO

des_crypt(3), **publickey**(5)

NAME

yp – NIS protocol

PROTOCOL

/usr/include/rpcsvc/yp.x

DESCRIPTION

The Network Information Service (NIS) is used for the administration of network-wide databases. The service is composed mainly of two programs: **YPBINDPROG** for finding a NIS server and **YPPROG** for accessing the NIS databases.

PROGRAMMING

Refer to **ypclnt**(3N) for information on the programmatic interface to NIS servers and databases.

SEE ALSO

ypclnt(3N), **yppasswd**(3R)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yppasswd – update user password in NIS

PROTOCOL

/usr/include/rpcsvc/yppasswd.x

DESCRIPTION

The **yppasswd**() protocol is used to change a user's password entry in the Network Information Service (NIS) password database.

If *oldpass* is indeed the old user password, this routine replaces the password entry with *newpw*. It returns 0 if successful.

PROGRAMMING

```
#include <rpcsvc/yppasswd.h>

yppasswd(oldpass, newpw)
        char *oldpass
        struct passwd *newpw;
```

SEE ALSO

yppasswd(1), **yppasswdd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

intro – introduction to device drivers, protocols, and network interfaces

DESCRIPTION

This section describes device drivers, high-speed network interfaces, and protocols available under SunOS. The system provides drivers for a variety of hardware devices, such as disks, magnetic tapes, serial communication lines, mice and frame buffers, as well as virtual devices such as pseudo-terminals and windows. SunOS provides hardware support and a network interface for the 10-Megabit Ethernet, along with interfaces for the IP protocol family and a STREAMS-based Network Interface Tap (NIT) facility.

In addition to describing device drivers that are supported by the 4.3BSD operating system, this section contains subsections that describe:

- SunOS-specific device drivers, under '4S'.
- Protocol families, under '4F'.
- Protocols and raw interfaces, under '4P'.
- STREAMS modules, under '4M'.
- Network interfaces, under '4N'.

Configuration

The SunOS kernel can be configured to include or omit many of the device drivers described in this section. The CONFIG section of the manual page gives the line(s) to include in the kernel configuration file for each machine architecture on which a device is supported. If no specific architectures are indicated, the configuration syntax applies to all Sun systems.

The GENERIC kernel is the default configuration for SunOS. It contains all of the optional drivers for a given machine architecture. See **config**(8), for details on configuring a new SunOS kernel.

The manual page for a device driver may also include a DIAGNOSTICS section, listing error messages that the driver might produce. Normally, these messages are logged to the appropriate system log using the kernel's standard message-buffering mechanism (see **syslogd**(8)); they may also appear on the system console.

Ioctls

Various special functions, such as querying or altering the operating characteristics of a device, are performed by supplying appropriate parameters to the **ioctl**(2) system call. These parameters are often referred to as "ioctls." Ioctls for a specific device are presented in the manual page for that device. Ioctls that pertain to a class of devices are listed in a manual page with a name that suggests the class of device, and ending in 'io', such as **mtio**(4) for magnetic tape devices, or **dkio**(4S) for disk controllers. In addition, some ioctls operate directly on higher-level objects such as files, terminals, sockets, and streams:

- Ioctls that operate directly on files, file descriptors, and sockets are described in **filio**(4). Note: the **fcntl**(2V) system call is the primary method for operating on file descriptors as such, rather than on the underlying files. Also note that the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. Ioctls for a specific network interface are documented in the manual page for that interface.
- Ioctls for terminals, including pseudo-terminals, are described in **termio**(4). This manual page includes information about both the BSD **termios** structure, as well as the System V **termio** structure.
- Ioctls for STREAMS are described in **streamio**(4).

Devices Always Present

Device drivers present in every kernel include:

- The paging device; see **drum**(4).
- Drivers for accessing physical, virtual, and I/O space in memory; see **mem**(4S).
- The data sink; see **null**(4).

Terminals and Serial Communications Devices

Serial communication lines are normally supported by the terminal driver; see **tty**(4). This driver manages serial lines provided by communications drivers, such as those described in **mti**(4S) and **zs**(4S). The terminal driver also handles serial lines provided by virtual terminals, such as the Sun console monitor described in **console**(4S), and true pseudo-terminals, described in **pty**(4).

Disk Devices

Drivers for the following disk controllers provide standard block and raw interfaces under SunOS;

- SCSI controllers, in **sd**(4S),
- Xylogics 450 and 451 SMD controllers, in **xy**(4S),
- Xylogics 7053 SMD controllers, in **xd**(4S).

Ioctls to query or set a disk's geometry and partitioning are described in **dkio**(4S).

Magnetic Tape Devices

Magnetic tape devices supported by SunOS include those described in **ar**(4S), **tm**(4S), **st**(4S), and **xt**(4S). Ioctls for all tape-device drivers are described in **mtio**(4S).

Frame Buffers

Frame buffer devices include color frame buffers described in the **cg***(4S) manual pages, monochrome frame buffers described in the **bw***(4S) manual pages, graphics processor interfaces described in the **gp***(4S) manual pages, and an indirect device for the console frame buffer described in **fb**(4S). Ioctls for all frame-buffer devices are described in **fbio**(4S).

Miscellaneous Devices

Miscellaneous devices include the console keyboard described in **kbd**(4S), the console mouse described in **mouse**(4S), window devices described in **win**(4S), and the DES encryption-chip interface described in **des**(4S).

Network-Interface Devices

SunOS supports the 10-Megabit Ethernet as its primary network interface; see **ie**(4S) and **le**(4S) for details. However, a software loopback interface, **lo**(4) is also supported. General properties of these network interfaces are described in **if**(4N), along with the ioctls that operate on them.

Support for network routing is described in **routing**(4N).

Protocols and Protocol Families

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in **inet**(4F), is the primary protocol family primary supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the **AF_INET** address family when binding a socket; see **socket**(2) for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in **ip**(4P). This is the default protocol for SOCK_RAW type sockets within the **AF_INET** domain.
- The Transmission Control Protocol (TCP); see **tcp**(4P). This is the default protocol for SOCK_STREAM type sockets.
- The User Datagram Protocol (UDP); see **udp**(4P). This is the default protocol for SOCK_DGRAM type sockets.
- The Address Resolution Protocol (ARP); see **arp**(4P).
- The Internet Control Message Protocol (ICMP); see **icmp**(4P).

The Network Interface Tap (NIT) protocol, described in **nit**(4P), is a STREAMS-based facility for accessing the network at the link level.

SEE ALSO

fcntl(2V), **getsockopt**(2), **ioctl**(2), **socket**(2), **ar**(4S), **arp**(4P), **dkio**(4S), **drum**(4), **fb**(4S), **fbio**(4S), **filio**(4), **icmp**(4P), **if**(4N), **inet**(4F), **ip**(4P), **kbd**(4S), **le**(4S), **lo**(4), **mem**(4S), **mti**(4S), **mtio**(4), **nit**(4P), **null**(4), **pty**(4), **routing**(4N), **sd**(4S), **st**(4S) **streamio**(4), **tcp**(4P), **termio**(4), **tm**(4S), **tty**(4), **udp**(4P), **win**(4S), **xd**(4S), **xy**(4S), **zs**(4S)

LIST OF DEVICES, INTERFACES AND PROTOCOLS

Name	Appears on Page	Description
alm	**mcp**(4S)	ALM-2 Asynchronous Line Multiplexer
ar	**ar**(4S)	Archive 1/4 inch Streaming Tape Drive
arp	**arp**(4P)	Address Resolution Protocol
atbus	**mem**(4S)	main memory and bus I/O space
audio	**audio**(4)	telephone quality audio device
bwtwo	**bwtwo**(4S)	black and white memory frame buffer
cdromio	**cdromio**(4S)	CDROM control operations
cgeight	**cgeight**(4S)	24-bit color memory frame buffer
cgfour	**cgfour**(4S)	Sun-3 color memory frame buffer
cgnine	**cgnine**(4S)	24-bit VME color memory frame buffer
cgsix	**cgsix**(4S)	accelerated 8-bit color frame buffer
cgthree	**cgthree**(4S)	8-bit color memory frame buffer
cgtwo	**cgtwo**(4S)	color graphics interface
console	**console**(4S)	console driver and terminal emulator
db	**db**(4M)	SunDials STREAMS module
des	**des**(4S)	DES encryption chip interface
dkio	**dkio**(4S)	generic disk control operations
drum	**drum**(4)	paging device
eeprom	**mem**(4S)	main memory and bus I/O space
fb	**fb**(4S)	driver for Sun console frame buffer
fbio	**fbio**(4S)	frame buffer control operations
fd	**fd**(4S)	Disk driver for Floppy Disk Controllers
filio	**filio**(4)	ioctls that operate directly on files, file descriptors, and sockets
fpa	**fpa**(4S)	Sun-3 floating-point accelerator
gpone	**gpone**(4S)	graphics processor
icmp	**icmp**(4P)	Internet Control Message Protocol
ie	**ie**(4S)	Intel 10 Mb/s Ethernet interface
if	**if**(4N)	general properties of network interfaces
inet	**inet**(4F)	Internet protocol family
ip	**ip**(4P)	Internet Protocol
kb	**kb**(4M)	Sun keyboard STREAMS module
kbd	**kbd**(4S)	Sun keyboard
kmem	**mem**(4S)	main memory and bus I/O space
ldterm	**ldterm**(4M)	standard terminal STREAMS module
le	**le**(4S)	LANCE 10Mb/s Ethernet interface
lo	**lo**(4N)	software loopback network interface
lofs	**lofs**(4S)	loopback virtual file system
mcp	**mcp**(4S)	MCP Multiprotocol Communications Processor
mem	**mem**(4S)	main memory and bus I/O space
mouse	**mouse**(4S)	Sun mouse
ms	**ms**(4M)	Sun mouse STREAMS module
mti	**mti**(4S)	Systech MTI-800/1600 multi-terminal interface
mtio	**mtio**(4)	general magnetic tape interface

NFS	**nfs**(4P)	network file system
nit	**nit**(4P)	Network Interface Tap
nit_buf	**nit_buf**(4M)	STREAMS NIT buffering module
nit_if	**nit_if**(4M)	STREAMS NIT device interface module
nif_pf	**nit_pf**(4M)	STREAMS NIT packet filtering module
null	**null**(4)	data sink
openprom	**openprom**(4S)	PROM monitor configuration interface
pp	**pp**(4)	Centronics-compatible parallel printer port
pty	**pty**(4)	pseudo-terminal driver
rfs	**rfs**(4)	remote file sharing service
root	**root**(4S)	pseudo-driver for Sun386i root disk
routing	**routing**(4N)	system supporting for local network packet routing
sbus	**mem**(4S)	main memory and bus I/O space
sd	**sd**(4S)	driver for SCSI disk devices
sockio	**sockio**(4)	ioctls that operate directly on sockets
sr	**sr**(4S)	driver for CDROM SCSI controller
st	**st**(4S)	driver for SCSI tape devices
streamio	**streamio**(4)	STREAMS ioctl commands
taac	**taac**(4S)	Sun applications accelerator
tcp	**tcp**(4P)	Internet Transmission Control Protocol
tcptli	**tcptli**(4P)	TLI-Conforming TCP Stream-Head
termio	**termio**(4)	general terminal interface
tfs	**tfs**(4S)	translucent file service
tm	**tm**(4S)	Tapemaster 1/2 inch tape controller
tmpfs	**tmpfs**(4S)	memory based filesystem
ttcompat	**ttcompat**(4M)	V7 and 4BSD STREAMS compatibility module
tty	**tty**(4)	controlling terminal interface
udp	**udp**(4P)	Internet User Datagram Protocol
unix	**unix**(4F)	UNIX domain protocol family
vd	**vd**(4)	loadable modules interface
vme16d16	**mem**(4S)	main memory and bus I/O space
vme16d32	**mem**(4S)	main memory and bus I/O space
vme24d16	**mem**(4S)	main memory and bus I/O space
vme24d32	**mem**(4S)	main memory and bus I/O space
vme32d16	**mem**(4S)	main memory and bus I/O space
vme32d32	**mem**(4S)	main memory and bus I/O space
vpc	**vpc**(4S)	Systech VPC-2200 Versatec printer/plotter
win	**win**(4S)	Sun window system
xd	**xd**(4S)	Disk driver for Xylogics 7053 SMD Disk Controller
xt	**xt**(4S)	Xylogics 472 1/2 inch tape controller
xy	**xy**(4S)	Disk driver for Xylogics 450 and 451 SMD Disk Controller
zero	**mem**(4S)	main memory and bus I/O space
zero	**zero**(4S)	source of zeroes
zs	**zs**(4S)	Zilog 8530 SCC serial communications driver

NAME

ar – Archive 1/4 inch Streaming Tape Drive

AVAILABILITY

Sun-3 and Sun-4 systems only.

DESCRIPTION

The Archive tape controller is a Sun 'QIC-II' interface to an Archive streaming tape drive. It provides a standard tape interface to the device, see **mtio**(4), with some deficiencies listed under BUGS below.

The maximum blocksize for the raw device is limited only by available memory.

FILES

/dev/rar*
/dev/nrar*　　non-rewinding

SEE ALSO

mtio(4)

DIAGNOSTICS

ar*: would not initialize

ar*: already open
The tape can be opened by only one process at a time

ar*: no such drive

ar*: no cartridge in drive

ar*: cartridge is write protected

ar: interrupt from unitialized controller %x

ar*: many retries, consider retiring this

ar*: %b error at block #

ar*: %b error at block #

ar: giving up on Rdy, try

BUGS

The tape cannot reverse direction so the **BSF** and **BSR** ioctls are not supported.

The **FSR** ioctl is not supported.

The system will hang if the tape is removed while running.

When using the raw device, the number of bytes in any given transfer must be a multiple of 512 bytes. If it is not, the device driver returns an error.

The driver will only write an EOF mark on close if the last operation was a write, without regard for the mode used when opening the file. This delete empty files on a raw tape copy operation.

NAME

arp – Address Resolution Protocol

CONFIG

pseudo-device ether

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

DESCRIPTION

ARP is a protocol used to dynamically map between Internet Protocol (IP) and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers. It is not specific to the Internet Protocol or to the 10Mb/s Ethernet, but this implementation currently supports only that combination.

ARP caches IP-to-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

To facilitate communications with systems which do not use ARP, **ioctl()** requests are provided to enter and delete entries in the IP-to-Ethernet tables.

USAGE

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
struct arpreq arpreq;
ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each **ioctl()** takes the same structure as an argument. **SIOCSARP** sets an ARP entry, **SIOCGARP** gets an ARP entry, and **SIOCDARP** deletes an ARP entry. These **ioctl()** requests may be applied to any socket descriptor *s*, but only by the super-user. The **arpreq** structure contains:

```
/*
 * ARP ioctl request
 */
struct arpreq {
        struct sockaddr  arp_pa;        /* protocol address */
        struct sockaddr  arp_ha;        /* hardware address */
        int      arp_flags;             /* flags */
};
/*  arp_flags field values */
#define ATF_COM              0x2        /* completed entry (arp_ha valid) */
#define ATF_PERM             0x4        /* permanent entry */
#define ATF_PUBL             0x8        /* publish (respond for other host) */
#define ATF_USETRAILERS                 0x10    /* send trailer packets to host */
```

The address family for the **arp_pa** sockaddr must be **AF_INET**; for the **arp_ha** sockaddr it must be **AF_UNSPEC**. The only flag bits which may be written are **ATF_PERM**, **ATF_PUBL** and **ATF_USETRAILERS**. **ATF_PERM** makes the entry permanent if the **ioctl()** call succeeds. The peculiar nature of the ARP tables may cause the **ioctl()** to fail if more than 6 (permanent) IP addresses hash to the same slot. **ATF_PUBL** specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an "ARP server" which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts which wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The **ATF_USETRAILERS** flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host's address).

SEE ALSO

ec(4S), **ie**(4S), **inet**(4F), **arp**(8C), **ifconfig**(8C)

Plummer, Dave, ''*An Ethernet Address Resolution Protocol -or- Converting Network Protocol Addresses to 48.bit Ethernet Addresses for Transmission on Ethernet Hardware*,'' RFC 826, Network Information Center, SRI International, Menlo Park, Calif., November 1982. (Sun 800-1059-10)

Leffler, Sam, and Michael Karels, ''*Trailer Encapsulations*,'' RFC 893, Network Information Center, SRI International, Menlo Park, Calif., April 1984.

DIAGNOSTICS

duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x.

ARP has discovered another host on the local network which responds to mapping requests for its own Internet address.

BUGS

ARP packets on the Ethernet use only 42 bytes of data, however, the smallest legal Ethernet packet is 60 bytes (not including CRC). Some systems may not enforce the minimum packet size, others will.

NAME

audio – telephone quality audio device

CONFIG

device-driver audio

AVAILABILITY

This device is available with SPARCstation 1 systems only.

DESCRIPTION

The **audio** device plays and records a single channel of sound using the AM79C30A Digital Subscriber Controller chip. The chip has a built-in analog to digital converter (ADC) and digital to analog converter (DAC) that can drive either the built-in speaker or an external headphone jack, selectable under software control. Digital audio data is sampled at a rate of 8000 samples per second with 12-bit precision, though the data is compressed, using *u*–law encoding, to 8-bit samples. The resulting audio data quality is equivalent to that of standard telephone service.

The **audio** driver is implemented as a STREAMS device. In order to record audio input, applications **open**(2V) the **/dev/audio** device and read data from it using the **read**(2V) system call. Similarly, sound data is queued to the audio output port by using the **write**(2V) system call.

Opening the Audio Device

The audio device is treated as an exclusive resource: only one process may typically open the device at a time. However, two processes may simultaneously access the device if one opens it read-only and the other opens it write-only.

When a process cannot open **/dev/audio** because the requested access mode is busy:

- if the **O_NDELAY** flag is set in the **open()** *flags* argument, then **open()** returns −1 immediately, with *errno* set to EBUSY.
- if **O_NDELAY** is not set, then **open()** hangs until the device is available or a signal is delivered to the process, in which case **open()** returns −1 with *errno* set to EINTR.

Since the audio device grants exclusive read or write access to a single process at a time, long-lived audio applications may choose to close the device when they enter an idle state, reopening it when required. The *play.waiting* and *record.waiting* flags in the audio information structure (see below) provide an indication that another process has requested access to the device. This information is advisory only; background audio output processes, for example, may choose to relinquish the audio device whenever another process requests write access.

Recording Audio Data

The **read()** system call copies data from the system buffers to the application. Ordinarily, **read()** blocks until the user buffer is filled. The **FIONREAD ioctl** (see **filio**(4)) may be used to determine the amount of data that may be read without blocking. The device may alternatively be set to a non-blocking mode, in which case **read()** completes immediately, but may return fewer bytes than requested. Refer to the **read**(2V) manual page for a complete description of this behavior.

When the audio device is opened with read access, the device driver immediately starts buffering audio input data. Since this consumes system resources, processes that do not record audio data should open the device write-only (**O_WRONLY**).

The transfer of input data to STREAMS buffers may be paused (or resumed) by using the **AUDIO_SETINFO ioctl** to set (or clear) the *record.pause* flag in the audio information structure (see below). All unread input data in the STREAMS queue may be discarded by using the **I_FLUSH** STREAMS **ioctl** (see **streamio**(4)).

Input data accumulates in STREAMS buffers at a rate of 8000 bytes per second. If the application that consumes the data cannot keep up with this data rate, the STREAMS queue may become full. When this occurs, the *record.error* flag is set in the audio information structure and input sampling ceases until there is room in the input queue for additional data. In such cases, the input data stream contains a discontinuity. For this reason, audio recording applications should open the audio device when they are prepared to begin reading data, rather than at the start of extensive initialization.

Playing Audio Data

The **write( )** system call copies data from an applications buffer to the STREAMS output queue. Ordinarily, **write( )** blocks until the entire user buffer is transferred. The device may alternatively be set to a non-blocking mode, in which case **write( )** completes immediately, but may have transferred fewer bytes than requested (see **write**(2V)).

Although **write( )** returns when the data is successfully queued, the actual completion of audio output may take considerably longer. The **AUDIO_DRAIN ioctl** may be issued to allow an application to block until all of the queued output data has been played. Alternatively, a process may request asynchronous notification of output completion by writing a zero-length buffer (end-of-file record) to the output stream. When such a buffer has been processed, the *play.eof* flag in the audio information structure (see below) is incremented.

The final **close( )** of the file descriptor hangs until audio output has drained. If a signal interrupts the **close( )** , or if the process exits without closing the device, any remaining data queued for audio output is flushed and the device is closed immediately.

The conversion of output data may be paused (or resumed) by using the **AUDIO_SETINFO ioctl** to set (or clear) the *play.pause* flag in the audio information structure. Queued output data may be discarded by using the **I_FLUSH** STREAMS **ioctl**.

Output data is played from the STREAMS buffers at a rate of 8000 bytes per second. If the output queue becomes empty, the *play.error* flag is set in the audio information structure and output ceases until additional data is written.

Asynchronous I/O

The **I_SETSIG** STREAMS **ioctl** may be used to enable asynchronous notification, via the **SIGPOLL** signal, of input and output ready conditions. This, in conjunction with non-blocking **read( )** and **write( )** requests, is normally sufficient for applications to maintain an audio stream in the background. Alternatively, asynchronous reads and writes may be initiated using the **aioread**(3) functions.

Audio Data Encoding

The data samples processed by the audio device are encoded in 8 bits. The high-order bit is a sign bit: 1 represents positive data and 0 represents negative data. The low-order 7 bits represent signal magnitude and are inverted (1's complement). The magnitude is encoded according to a *u*-law transfer function; such an encoding provides an improved signal-to-noise ratio at low amplitude levels. In order to achieve best results, the audio recording gain should be set so that typical amplitude levels lie within approximately three-fourths of the full dynamic range.

Audio Control Pseudo-Device

It is sometimes convenient to have an application, such as a volume control panel, modify certain characteristics of the audio device while it is being used by an unrelated process. The **/dev/audioctl** minor device is provided for this purpose. Any number of processes may open **/dev/audioctl** simultaneously. However, **read( )** and **write( )** system calls are ignored by **/dev/audioctl**. The **AUDIO_GETINFO** and **AUDIO_SETINFO ioctl** commands may be issued to **/dev/audioctl** in order to determine the status or alter the behavior of **/dev/audio**.

Audio Status Change Notification

Applications that open the audio control pseudo-device may request asynchronous notification of changes in the state of the audio device by setting the **S_MSG** flag in an **I_SETSIG** STREAMS **ioctl**. Such processes receive a **SIGPOLL** signal when any of the following events occurs:

- An **AUDIO_SETINFO ioctl** has altered the device state.
- An input overflow or output underflow has occurred.
- An end-of-file record (zero-length buffer) has been processed on output.
- An **open( )** or **close( )** of **/dev/audio** has altered the device state.

Audio Information Structure

The state of the audio device may be polled or modified using the **AUDIO_GETINFO** and **AUDIO_SETINFO ioctl** commands. These commands operate on the **audio_info** structure, defined in **<sun/audioio.h>** as follows:

```
/* Data encoding values, used below in the encoding field */
#define AUDIO_ENCODING_ULAW      (1)     /* u-law encoding */
#define AUDIO_ENCODING_ALAW      (2)     /* A-law encoding */

/* These ranges apply to record, play, and monitor gain values */
#define AUDIO_MIN_GAIN       (0)         /* minimum gain value */
#define AUDIO_MAX_GAIN       (255)       /* maximum gain value */

/* Audio I/O channel status, used below in the audio_info structure */
struct audio_prinfo {
        /* The following values describe the audio data encoding */
        unsigned            sample_rate;    /* samples per second */
        unsigned            channels;       /* number of interleaved channels */
        unsigned            precision;      /* number of bits per sample */
        unsigned            encoding;       /* data encoding method */

        /* The following values control audio device configuration */
        unsigned            gain;           /* gain level */
        unsigned            port;           /* selected I/O port */

        /* The following values describe the current device state */
        unsigned            samples;        /* number of samples converted */
        unsigned            eof;            /* End Of File counter (play only) */
        unsigned char       pause;          /* non-zero if paused, zero to resume */
        unsigned char       error;          /* non-zero if overflow/underflow */
        unsigned char       waiting;        /* non-zero if a process wants access */

        /* The following values are read-only device state flags */
        unsigned char       open;           /* non-zero if open access granted */
        unsigned char       active;         /* non-zero if I/O active */
};

/* This structure is used in AUDIO_GETINFO and AUDIO_SETINFO ioctl commands */
typedef struct audio_info {
        struct audio_prinfo    record;         /* input status information */
        struct audio_prinfo    play;           /* output status information */
        unsigned               monitor_gain;   /* input to output mix */
} audio_info_t;
```

The *play.gain* and *record.gain* fields specify the output and input volume levels. A value of **AUDIO_MAX_GAIN** indicates maximum gain. The device also allows input data to be monitored by mixing audio input onto the output channel. The *monitor_gain* field controls the level of this feedback path. The *play.port* field controls the output path for the audio device. It may be set to either **AUDIO_SPEAKER** or **AUDIO_HEADPHONE** to direct output to the built-in speaker or the headphone jack, respectively.

The *play.pause* and *record.pause* flags may be used to pause and resume the transfer of data between the audio device and the STREAMS buffers. The *play.error* and *record.error* flags indicate that data underflow or overflow has occurred. The *play.active* and *record.active* flags indicate that data transfer is currently active in the corresponding direction.

The *play.open* and *record.open* flags indicate that the device is currently open with the corresponding access permission. The *play.waiting* and *record.waiting* flags provide an indication that a process may be waiting to access the device. These flags are set automatically when a process blocks on **open()**, though they may also be set using the **AUDIO_SETINFO ioctl** command. They are cleared only when a process relinquishes access by closing the device.

The *play.samples* and *record.samples* fields are initialized, at **open**( ), to zero and increment each time a data sample is copied to or from the associated STREAMS queue. Applications that keep track of the number of samples read or written may use these fields to determine exactly how many samples remain in the STREAMS buffers. The *play.eof* field increments whenever a zero-length output buffer is synchronously processed. Applications may use this field to detect the completion of particular segments of audio output.

The *sample_rate*, *channels*, *precision*, and *encoding* fields report the audio data format in use by the device. For now, these values are read-only; however, future audio device implementations may support more than one data encoding format, in which case applications might be able to modify these fields.

Filio and STREAMS IOCTLS

All of the **filio**(4) and **streamio**(4) **ioctl** commands may be issued for the **/dev/audio** device. Because the **/dev/audioctl** device has its own STREAMS queues, most of these commands neither modify nor report the state of **/dev/audio** if issued for the **/dev/audioctl** device. The **I_SETSIG ioctl** may be issued for **/dev/audioctl** to enable the notification of audio status changes, as described above.

Audio IOCTLS

The audio device additionally supports the following **ioctl** commands:

AUDIO_DRAIN

The argument is ignored. This command suspends the calling process until the output STREAMS queue is empty, or until a signal is delivered to the calling process. It may only be issued for the **/dev/audio** device. An implicit **AUDIO_DRAIN** is performed on the final **close**( ) of **/dev/audio**.

AUDIO_GETINFO

The argument is a pointer to an **audio_info** structure. This command may be issued for either **/dev/audio** or **/dev/audioctl**. The current state of the **/dev/audio** device is returned in the structure.

AUDIO_SETINFO

The argument is a pointer to an **audio_info** structure. This command may be issued for either **/dev/audio** or **/dev/audioctl**. This command configures the audio device according to the structure supplied and overwrites the structure with the new state of the device. [Note: The *play.samples*, *record.samples*, *play.error*, *record.error*, and *play.eof* fields are modified to reflect the state of the device when the **AUDIO_SETINFO** was issued. This allows programs to atomically modify these fields while retrieving the previous value.]

Certain fields in the information structure, such as the *pause* flags, are treated as read-only when **/dev/audio** is not open with the corresponding access permission. Other fields, such as the gain levels and encoding information, may have a restricted set of acceptable values. Applications that attempt to modify such fields should check the returned values to be sure that the corresponding change took effect.

Once set, the following values persist through subsequent **open**( ) and **close**( ) calls of the device: *play.gain*, *record.gain*, *monitor_gain*, *play.port*, and *record.port*. All other state is reset when the corresponding I/O stream of **/dev/audio** is closed.

The **audio_info** structure may be initialized through the use of the **AUDIO_INITINFO** macro. This macro sets all fields in the structure to values that are ignored by the **AUDIO_SETINFO** command. For instance, the following code switches the output port from the built-in speaker to the headphone jack without modifying any other audio parameters:

```
audio_info_t      info;

AUDIO_INITINFO(&info);
info.play.port = AUDIO_HEADPHONE;
err = ioctl(audio_fd, AUDIO_SETINFO, &info);
```

This technique is preferred over using a sequence of **AUDIO_GETINFO** followed by **AUDIO_SETINFO**.

Unsupported Device Control Features

The AM79C30A chip is capable of a performing a number of functions that are not currently supported by the device driver, many of which were designed primarily for telephony applications. For example, the chip can generate ringer tones and has a number of specialized filtering capabilities that are designed to compensate for different types of external speakers and microphones.

Ordinarily, applications do not need to access these capabilities and, further, altering the chip's characteristics may interfere with its normal behavior. However, knowledgeable applications may use the unsupported **AUDIOGETREG** and **AUDIOSETREG** **ioctl** commands to read and write the chip registers directly. The description of this interface may be found in **<sbusdev/audio_79C30.h>**. Note: these commands are supplied for prototyping purposes only and may become obsolete in a future release of the audio driver.

FILES

/dev/audio
/dev/audioctl
/usr/demo/SOUND

SEE ALSO

ioctl(2), **poll**(2), **read**(2V), **write**(2V), **aioread**(3), **filio**(4), **streamio**(4)

AMD data sheet for the AM79C30A Digital Subscriber Controller, Publication number 09893.

BUGS

Due to a *feature* of the STREAMS implementation, programs that are terminated or exit without closing the **audio** device may hang for a short period while audio output drains. In general, programs that produce audio output should catch the **SIGINT** signal and flush the output stream before exiting.

The current driver implementation does not support the A-law encoding mode of the AM79C30A chip. Future implementations may permit the **AUDIO_SETINFO** **ioctl** to modify the *play.encoding* and *record.encoding* fields of the device information structure to enable this mode.

FUTURE DIRECTIONS

Workstation audio resources should be managed by a networked audio server, in the same way that the video monitor is manipulated by a window system server. For the time being, we encourage you to write your programs in a modular fashion, isolating the **audio** device-specific functions, so that they may be easily ported to such an environment.

NAME

bwtwo – black and white memory frame buffer

CONFIG — SUN-3, SUN-3x SYSTEMS

device bwtwo0 at obmem 1 csr 0xff000000 priority 4
device bwtwo0 at obmem 2 csr 0x100000 priority 4
device bwtwo0 at obmem 3 csr 0xff000000 priority 4
device bwtwo0 at obmem 4 csr 0xff000000
device bwtwo0 at obmem 7 csr 0xff000000 priority 4
device bwtwo0 at obmem ? csr 0x50300000 priority 4

The first synopsis line given above is used to generate a kernel for Sun-3/75, Sun-3/140 or Sun-3/160 systems; the second, for a Sun-3/50 system; the third, for a Sun-3/260 system; the fourth, for a Sun-3/110 system; the fifth, for a Sun-3/60 system; and the sixth for Sun-3/80 and Sun-3/470 systems.

CONFIG — SUN-4 SYSTEMS

device bwtwo0 at obio 1 csr 0xfd000000 priority 4
device bwtwo0 at obio 2 csr 0xfb300000 priority 4
device bwtwo0 at obio 3 csr 0xfb300000 priority 4
device bwtwo0 at obio 4 csr 0xfb300000 priority 4

The first synopsis line given above should be used to generate a kernel for a Sun-4/260 or Sun-4/280 system; the second, for a Sun-4/110 system; the third for a Sun-4/330 system; and the fourth for a Sun-4/460 system.

CONFIG — SPARCstation 1 SYSTEMS

device-driver bwtwo

CONFIG — Sun386i SYSTEM

device bwtwo0 at obmem ? csr 0xA0200000

DESCRIPTION

The **bwtwo** interface provides access to Sun monochrome memory frame buffers. It supports the ioctls described in **fbio**(4S).

If **flags 0x1** is specified, frame buffer write operations are buffered through regular high-speed RAM. This "copy memory" mode of operation speeds frame buffer accesses, but consumes an extra 128K bytes of memory. Only Sun-3/75, Sun-3/140, and Sun-3/160 systems support copy memory; on other systems a warning message is printed and the flag is ignored.

Reading or writing to the frame buffer is not allowed — you must use the **mmap**(2) system call to map the board into your address space.

FILES

/dev/bwtwo[0-9] device files

SEE ALSO

mmap(2), **cgfour**(4S), **fb**(4S), **fbio**(4S)

BUGS

Use of vertical-retrace interrupts is not supported.

NAME

cdromio – CDROM control operations

DESCRIPTION

The Sun CDROM device driver supports a set of **ioctl**(2) commands for audio operations and CDROM specific operations. It also supports the **dkio**(4S) operations — generic disk control operation for all Sun disk drivers. See **dkio(4S)** Basic to these **cdromio ioctl()** requests are the definitions in **<scsi/targets/srdef.h>** or **<sundev/srreg.h>**

```
/*
 * CDROM I/O controls type definitions
 */

/* definition of play audio msf structure */
struct cdrom_msf {
        unsigned char   cdmsf_min0;     /* starting minute */
        unsigned char   cdmsf_sec0;     /* starting second */
        unsigned char   cdmsf_frame0;   /* starting frame  */
        unsigned char   cdmsf_min1;     /* ending minute   */
        unsigned char   cdmsf_sec1;     /* ending second   */
        unsigned char   cdmsf_frame1;   /* ending frame    */
};

/* definition of play audio track/index structure */
struct cdrom_ti {
        unsigned char   cdti_trk0;      /* starting track */
        unsigned char   cdti_ind0;      /* starting index */
        unsigned char   cdti_trk1;      /* ending track */
        unsigned char   cdti_ind1;      /* ending index */
};

/* definition of read toc header structure */
struct cdrom_tochdr {
        unsigned char   cdth_trk0;      /* starting track */
        unsigned char   cdth_trk1;      /* ending track */
};

/* definition of read toc entry structure */
struct cdrom_tocentry {
        unsigned char   cdte_track;
        unsigned char   cdte_adr        :4;
        unsigned char   cdte_ctrl       :4;
        unsigned char   cdte_format;
        union {
                struct {
                        unsigned char   minute;
                        unsigned char   second;
                        unsigned char   frame;
                } msf;
                int     lba;
        } cdte_addr;
        unsigned char   cdte_datamode;
};
```

```
/*
 * Bitmask for CDROM data track in the cdte_ctrl field
 * A track is either data or audio.
 */
#define CDROM_DATA_TRACK   0x04

/*
 * CDROM address format definition, for use with struct cdrom_tocentry
 */
#define CDROM_LBA      0x01
#define CDROM_MSF      0x02

/*
 * For CDROMREADTOCENTRY, set the cdte_track to CDROM_LEADOUT to get
 * the information for the leadout track.
 */
#define CDROM_LEADOUT        0xAA

struct cdrom_subchnl {
        unsigned char   cdsc_format;
        unsigned char   cdsc_audiostatus;
        unsigned char   cdsc_adr:       4;
        unsigned char   cdsc_ctrl:      4;
        unsigned char   cdsc_trk;
        unsigned char   cdsc_ind;
        union {
                struct {
                        unsigned char   minute;
                        unsigned char   second;
                        unsigned char   frame;
                } msf;
                int     lba;
        } cdsc_absaddr;
        union {
                struct {
                        unsigned char   minute;
                        unsigned char   second;
                        unsigned char   frame;
                } msf;
                int     lba;
        } cdsc_reladdr;
};

/*
 * Definition for audio status returned from Read Sub-channel
 */
#define CDROM_AUDIO_INVALID    0x00  /* audio status not supported */
#define CDROM_AUDIO_PLAY       0x11  /* audio play operation in progress */
#define CDROM_AUDIO_PAUSED     0x12  /* audio play operation paused */
#define CDROM_AUDIO_COMPLETED  0x13  /* audio play successfully completed */
#define CDROM_AUDIO_ERROR      0x14  /* audio play stopped due to error */
#define CDROM_AUDIO_NO_STATUS  0x15  /* no current audio status to return */
```

```
/* definition of audio volume control structure */
struct cdrom_volctrl {
        unsigned char   cdvc_chnl0;
        unsigned char   cdvc_chnl1;
        unsigned char   cdvc_chnl2;
        unsigned char   cdvc_chnl3;
};

struct cdrom_read {
    int    cdread_lba;
    caddr_t cdread_bufaddr;
    int    cdread_buflen;
};

#define CDROM_MODE1_SIZE     2048
#define CDROM_MODE2_SIZE     2336

/*
 * CDROM I/O control commands
 */
#define CDROMPAUSE   _IO(c, 10)  /* Pause Audio Operation */

#define CDROMRESUME  _IO(c, 11)         /* Resume paused Audio Operation */

#define CDROMPLAYMSF _IOW(c, 12, struct cdrom_msf)    /* Play Audio MSF */

#define CDROMPLAYTRKIND  _IOW(c, 13, struct cdrom_ti)  /* Play Audio Trk/ind */

#define CDROMREADTOCHDR  _IOR(c, 103, struct cdrom_tochdr) /* Read TOC hdr */

#define CDROMREADTOCENTRY _IOWR(c, 104, struct cdrom_tocentry) /* Read TOC */

#define CDROMSTOP       _IO(c, 105)         /* Stop the cdrom drive */

#define CDROMSTART      _IO(c, 106)         /* Start the cdrom drive */

#define CDROMEJECT      _IO(c, 107)         /* Ejects the cdrom caddy */

#define CDROMVOLCTRL    _IOW(c, 14, struct cdrom_volctrl) /* volume control */

#define CDROMSUBCHNL  _IOWR(c, 108, struct cdrom_subchnl) /* read subchannel */

#define CDROMREADMODE2        _IOW(c, 110, struct cdrom_read) /* mode 2 */

#define CDROMREADMODE1        _IOW(c, 111, struct cdrom_read) /* mode 1 */
```

The **CDROMPAUSE ioctl()** pauses the current audio play operation and the **CDROMRESUME ioctl()** resumes the paused audio play operation. The **CDROMSTART ioctl()** spins up the disc and seeks to the last address requested, while the **CDROMSTOP ioctl()** spins down the disc and the **CDROMEJECT ioctl()** ejects the caddy with the disc. All of the above **ioctl()** calls only take a file descriptor and a command as arguments. They have the form:

```
        ioctl(fd, cmd)
                int     fd;
                int     cmd;
```

The rest of the **ioctl**( ) calls have the form:

```
ioctl(fd, cmd, ptr)
        int     fd;
        int     cmd;
        char    *ptr;
```

where *ptr* is a pointer to a struct or an integer.

The **CDROMPLAYMSF ioctl**( ) command requests the drive to output the audio signals staring at the specified starting address and continue the audio play until the specified ending address is detected. The address is in **MSF** (minute, second, frame) format. The third argument of the function call is a pointer to the type **struct cdrom_msf.**

The **CDROMPLAYTRKIND ioctl**( ) command is similar to **CDROMPLAYMSF**. The starting and ending address is in track/index format. The third argument of the function call is a pointer to the type **struct cdrom_ti.**

The **CDROMREADTOCHDR ioctl**( ) command returns the header of the **TOC** (table of contents). The header consists of the starting tracking number and the ending track number of the disc. These two numbers are returned through a pointer of **struct cdrom_tochdr**. While the disc can start at any number, all tracks between the first and last tracks are in contiguous ascending order. A related **ioctl**( ) command is **CDROMREADTOCENTRY**. This command returns the information of a specified track. The third argument of the function call is a pointer to the type **struct cdrom_tocentry**. The caller need to supply the track number and the address format. This command will return a 4-bit **adr** field, a 4-bit **ctrl** field, the starting address in **MSF** format or **LBA** format, and the data mode if the track is a data track. The **ctrl** field specifies whether the track is data or audio. To get information for the lead-out area, supply the **ioctl**( ) command with the track field set to **CDROM_LEADOUT** (0xAA).

The **CDROMVOLCTRL ioctl**( ) command controls the audio output level. The SCSI command allows the control of up to 4 channels. The current implementation of the supported **CDROM** drive only uses channel 0 and channel 1. The valid values of volume control are between 0x00 and 0xFF, with a value of 0xFF indicating maximum volume. The third argument of the function call is a pointer to **struct cdrom_volctrl** which contains the output volume values.

The **CDROMSUBCHNL ioctl**( ) command reads the Q sub-channel data of the current block. The sub-channel data includes track number, index number, absolute **CDROM** address, track relative **CDROM** address, control data and audio status. All information is returned through a pointer to **struct cdrom_subchnl**. The caller needs to supply the address format for the returned address.

The **CDROMREADMODE2** and **CDROMREADMODE1 ioctl**( ) commands are only available on SPARCstation 1 systems.

Finally, on SPARCstation 1 systems only, the driver supports the user SCSI command interface. By issuing the **ioctl**( ) command, **USCSICMD**, The caller can supply any **SCSI-2** commands that the **CDROM** drive supports. The caller has to provide all the parameters in the SCSI command block, as well as other information such as the user buffer address and buffer length. See the definitions in **<scsi/impl/uscsi.h>**. The **ioctl**( ) call has the form:

```
ioctl(fd, cmd, ptr)
        int     fd;
        int     cmd;
        char    *ptr;
```

where *ptr* is a pointer to the type:

```
struct uscsi_scmd {
        caddr_t uscsi_cdb;
        int     uscsi_cdblen;
        caddr_t uscsi_bufaddr;
        int     uscsi_buflen;
        unsigned char   uscsi_status;
        int     uscsi_flags;
};
```

uscsi_cdb is a pointer to the SCSI command block. Group 0 **cdb's** are 6 bytes long while the other groups are 10 bytes or 12 bytes. **uscsi_cdblen** is the length of the **cdb**. **uscsi_bufaddr** is the pointer to the user buffer for parameter passing or data input/output. *buflen* is the length of the user buffer. **uscsi_flags** are the execution flags for SCSI input/output. The possible flags are **USCSI_SILENT**, **USCSI_DIAGNOSE**, **USCSI_ISOLATE**, **USCSI_READ**, and **USCSI_WRITE**.

FILES

/usr/include/scsi/targets/srdef.h
/usr/include/scsi/impl/uscsi.h
/usr/include/sundev/srreg.h

SEE ALSO

ioctl(2), **dkio**(4S), **sr**(4S)

BUGS

The interface to this device is preliminary and subject to change in future releases. You are encouraged to write your programs in a modular fashion so that you can easily incorporate future changes.

NAME

cgeight – 24-bit color memory frame buffer

CONFIG — SUN-3 AND SUN-4 SYSTEMS

device cgeight0 at obmem 7 csr 0xff300000 priority 4
device cgeight0 at obio 4 csr 0xfb300000 priority 4

The first synopsis line should be used to generate a kernel for the Sun-3/60; the second synopsis for a Sun-4/110 or Sun-4/150 system.

CONFIG — SUN-3x SYSTEM

device cgeight0 at obio ? csr 0x50300000 priority 4

DESCRIPTION

The **cgeight** is a 24-bit color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented optionally on the Sun-4/110, Sun-4/150, Sun-3/60, Sun-3/470 and Sun-3/80 system models. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgeight** interface responds to two **cgeight**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **<sun/fbio.h>** as:

```
struct fbcmap {
        int             index;      /* first element (0 origin) */
        int             count;      /* number of elements */
        unsigned char   *red;       /* red color map elements */
        unsigned char   *green;     /* green color map elements */
        unsigned char   *blue;      /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The systems have an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX_GROUP** macro (see **<pixrect/pr_planegroups.h>**).

When using the **mmap** system call to map in the **cgeight** frame buffer. The device looks like:

DACBASE: 0x200000	**-> Brooktree Ramdac**	**16 bytes**
0x202000	**-> P4 Regiter**	**4 bytes**
OVLBASE: 0x210000	**-> Overlay Plane**	**1152x900x1**
0x230000	**-> Overlay Enable Planea**	**1152x900x1**
0x250000	**-> 24-bit Frame Buffera**	**1152x900x32**

FILES

/dev/cgeight0
<sun/fbio.h>
<pixrect/pr_planegroups.h>

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgfour – Sun-3 color memory frame buffer

CONFIG — SUN-3 SYSTEMS

device cgfour0 at obmem 4 csr 0xff000000 priority 4
device cgfour0 at obmem 7 csr 0xff300000 priority 4

The first synopsis line given should be used to generate a kernel for the Sun-3/110 system; and the second, for a Sun-3/60 system.

CONFIG — SUN-3x SYSTEMS

device cgfour0 at obmem ? csr 0x50300000 priority 4

CONFIG — SUN-4 SYSTEMS

device cgfour0 at obio 2 csr 0xfb300000 priority 4
device cgfour0 at obio 3 csr 0xfb300000 priority 4
device cgfour0 at obio 4 csr 0xfb300000 priority 4

The first synopsis line given should be used to generate a kernel for the Sun-4/110 system; the second, for a Sun-4/330 system; and the third for a Sun-4/460 system.

DESCRIPTION

The **cgfour** is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented on the Sun-3/110 system and some Sun-3/60 system models. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgfour** interface responds to two **cgfour**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **<sun/fbio.h>** as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The Sun-3/60 system has an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX_GROUP** macro (see **<pixrect/pr_planegroups.h>**).

FILES

/dev/cgfour0

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgnine – 24-bit VME color memory frame buffer

CONFIGURATION

device cgnine0 at vme32d32 ? csr 0x08000000 priority 4 vector cgnineintr 0xaa

DESCRIPTION

cgnine is a 24-bit double-buffered VME-based color frame buffer. It provides the standard frame buffer interface defined in **fbio**(4S), and and can be paired with the GP2 graphics accelerator board using **gpconfig**(8).

cgnine has two bits of overlay planes, each of which is a 1-bit deep frame buffer that overlays the 24-bit plane group. When either bit of the two overlay planes is non-zero, the pixel shows the color of the overlay plane. If both bits are zero, the color frame buffer underneath is visible.

The 24-bit frame buffer pixel is organized as one longword (32 bits) per pixel. The pixel format is defined in **<pixrect/pixrect.h>** as follows:

```
union fbunit {
unsigned int      packed; /* whole-sale deal */
struct {
        unsigned int     A:8;    /* unused, for now */
        unsigned int     B:8;    /* blue channel */
        unsigned int     G:8;    /* green channel */
        unsigned int     R:8;    /* red channel */
        }
                channel;         /* access per channel */
};
```

When the board is in double-buffer mode, the low 4 bits of each channel are ignored when written to, which yields 12-bit double-buffering.

The higher bit of the overlay planes ranges from offset 0 to 128K (0x20000) bytes. The lower bit ranges from 128K to 256K bytes. The 4MB (0x400000) of the 24-bit deep pixels begins at 256K. The addresses of the control registers start at the next page after the 24-bit deep pixels.

FILES

/dev/cgnine0	device special file
/dev/gpone0a	**cgnine** bound with GP2
/dev/fb	default frame buffer

SEE ALSO

mmap(2), **fbio**(4S), **gpone**(4S) **gpconfig**(8)

NAME

cgsix – accelerated 8-bit color frame buffer

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

device cgsix0 at obmem ? csr 0xff000000 priority 4
device cgsix0 at obmem ? csr 0x50000000 priority 4
device cgsix0 at obio ? csr 0xfb000000 priority 4

The first synopsis line given should be used for Sun-3/60 systems, the second for Sun-3x systems, and the third for Sun-4 systems.

CONFIG — SPARCstation 1 SYSTEMS

device-driver cgsix

DESCRIPTION

The **cgsix** is a low-end graphics accelerator designed to enhance vector and polygon drawing performance. It has an 8-bit color frame buffer and provides the standard frame buffer interface as defined in **fbio**(4S).

The cgsix has registers and memory that may be mapped with **mmap**(2), using the offsets defined in **<sundev/cg6reg.h>**.

FILES

/dev/cgsix0

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgthree – 8-bit color memory frame buffer

CONFIG — SPARCstation 1 SYSTEMS

device-driver cgthree

CONFIG — Sun386i SYSTEM

device cgthree0 at obmem ? csr 0xA0400000

AVAILABILITY

SPARCstation 1 and Sun386i systems only.

DESCRIPTION

cgthree is a color memory frame buffer. It provides the standard frame buffer interface as defined in **fbio**(4S).

FILES

/dev/cgthree[0-9]

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgtwo – color graphics interface

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

cgtwo0 at vme24d16 ? csr 0x400000 priority 4 vector cgtwointr 0xa8

DESCRIPTION

The **cgtwo** interface provides access to the color graphics controller board, which is normally supplied with a 19" 66 Hz non-interlaced color monitor. It provides the standard frame buffer interface as defined in **fbio**(4S).

The hardware consumes 4 megabytes of VME bus address space. The board starts at standard address 0x400000. The board must be configured for interrupt level 4.

FILES

/dev/cgtwo[0-9]

SEE ALSO

mmap(2), **fbio**(4S)

NAME

clone – open any minor device on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to **clone** during the open operation is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate stream to a previously unused minor device.

The **clone** driver supports only an **open**(2V) function. This open function performs all of the necessary work so that subsequent system calls (including **close**(2V)) require no further involvement of the **clone** driver.

ERRORS

clone generates an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

WARNINGS

Multiple opens of the same minor device are not supported through the **clone** interface. Executing **stat**(2V) on the file system node for a cloned device yields a different result than does executing **fstat** using a file descriptor obtained from opening that node.

SEE ALSO

close(2V), **open**(2V), **stat**(2V)

NAME

console – console driver and terminal emulator for the Sun workstation

CONFIG

None; included in standard system.

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/console", mode);

DESCRIPTION

console is an indirect driver for the Sun console terminal. On a Sun workstation, this driver refers to the workstation console driver, which implements a standard UNIX system terminal. On a Sun server without a keyboard or a frame buffer, this driver refers to the CPU serial port driver (**zs**(4S)); a terminal is normally connected to this port.

The workstation console does not support any of the **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure or by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

The workstation console driver calls the PROM resident monitor to output data to the console frame buffer. Keystrokes from the CPU serial port to which the keyboard is connected are routed through the keyboard STREAMS module (**kb**(4M)) and treated as input.

When the Sun window system **win**(4S) is active, console input is directed through the window system rather than being treated as input by the workstation console driver.

IOCTLS

An ioctl **TIOCCONS** can be applied to pseudo-terminals (**pty**(4)) to route output that would normally appear on the console to the pseudo-terminal instead. Thus, the window system does a **TIOCCONS** on a pseudo-terminal so that the system will route console output to the window to which that pseudo-terminal is connected, rather than routing output through the PROM monitor to the screen, since routing output through the PROM monitor destroys the integrity of the screen. Note: when you use **TIOCCONS** in this way, the console *input* is routed from the pseudo-terminal as well.

If a **TIOCCONS** is performed on **/dev/console**, or the pseudo-terminal to which console output is being routed is closed, output to the console will again be routed to the workstation console driver.

ANSI STANDARD TERMINAL EMULATION

The Sun Workstation's PROM monitor provides routines that emulates a standard ANSI X3.64 terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun terminal emulator and the VT100 are *not* compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with scrolling, (x, y) cursor addressability, and a number of other control functions.

The Sun console displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.

Control Sequence Syntax

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation

CTRL-*X*

for some character *X* , represents a control character.

Other ANSI control sequences are of the form

ESC [*params* char

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC represents the ASCII escape character (ESC, CTRL-[, 0x1B).

[The next character is a left square bracket '[' (0x5B).

params are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

char represents a function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character 'Escape'):

ESC[m	*select graphic rendition with default parameter*
ESC[7m	*select graphic rendition with reverse image*
ESC[33;54H	*set cursor position*
ESC[123;456;0;;3;B	*move cursor down*

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last *n* are used, where *n* is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as 'ESC[5M') is *not* equivalent to ESC[5;M (interpreted as 'ESC[5;1M') which is ultimately interpreted as 'ESC[1M').

In the syntax descriptions below, parameters are represented as '**#**' or '**#1;#2**'.

ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the SUNRESET escape sequence.

Control Character Functions

CTRL-G (0x7) Bell (BEL)
: The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It 'rings the bell' by flashing the entire screen. The window system flashes the window.

CTRL-H (0x8) Backspace (BS)
: The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

CTRL-I (0x9) Tab (TAB)
: The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

CTRL-J (0xA) Line-feed (LF)
: The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or ''wraps around'' depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, 'wrap-around' mode is entered. 'ESC [1 r' exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. When any line-feed occurs, the line that the cursor moves to is cleared. This means that no scrolling occurs. Wrap-around mode is not implemented in the window system.

The screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place and the console is in normal scroll mode ('ESC [1 r'), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when any control character from the set {**VT**, **FF**, **SO**, **SI**, **DLE**, **DC1**, **DC2**, **DC3**, **DC4**, **NAK**, **SYN**, **ETB**, **CAN**, **EM**, **SUB**, **ESC**, **FS**, **GS**, **RS**, **US**} is found. At that point, the screen is scrolled by N lines ($N \geq 1$) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the 'Set scrolling' (**ESC[r**) control function below.

CTRL-K (0xB) Reverse Line-feed
: The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.

CTRL-L (0xC) Form-feed (FF)
: The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.

CTRL-M (0xD) Return (CR)
: The cursor moves to the leftmost character position on the current line.

Escape Sequence Functions

CTRL-[(0x1B) Escape (ESC)
: This is the escape character. Escape initiates a multi-character control sequence.

ESC[#@ Insert Character (ICH)
: Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A Cursor Up (CUU)

Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B Cursor Down (CUD)

Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C Cursor Forward (CUF)

Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D Cursor Backward (CUB)

Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E Cursor Next Line (CNL)

Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f Horizontal And Vertical Position (HVP)

or

ESC[#1;#2H Cursor Position (CUP)

Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.

ESC[J Erase in Display (ED)

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K Erase in Line (EL)

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L Insert Line (IL)

Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with spaces; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M Delete Line (DL)

Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with spaces; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P Delete Character (DCH)

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC [#m Select Graphic Rendition (SGR)

Takes one parameter, # (default 0). Note: unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0 Normal rendition.

7 Negative (reverse) image.

Negative image displays characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and selects the negative image rendition.

ESC [p Black On White (SUNBOW)

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC [q White On Black (SUNWOB)

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC [#r Set scrolling (SUNSCRL)

Takes one parameter, # (default 0). Sets to # an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of "jump" when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates "wrap mode" instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever occurs. 'ESC [1 r' exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

ESC [s Reset terminal emulator (SUNRESET)

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are

4014 TERMINAL EMULATION

The PROM monitor for Sun models 100U and 150U provides the Sun Workstation with the capability to emulate a subset of the Tektronix 4014 terminal. This feature does not exist in other Sun PROMs and will be removed from models 100U and 150U in future Sun releases. **tektool**(1) provides Tektronix 4014 terminal emulation and should be used instead of relying on the capabilities of the PROM monitor.

FILES

/dev/console

SEE ALSO

tektool(1) **kb**(4M), **ldterm**(4M), **pty**(4), **termio**(4), **ttcompat**(4M), **win**(4S), **zs**(4S)

ANSI Standard X3.64, "*Additional Controls for Use with ASCII*", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

BUGS

TIOCCONS should be restricted to the owner of **/dev/console**.

NAME

db – SunDials STREAMS module

CONFIG

pseudo-device db

SYNOPSIS

```
#include <sys/stream.h>
#include <sundev/vuid_event.h>
#include <sundev/dbio.h>
#include <sys/time.h>
#include <sys/ioctl.h>
open("/dev/dialbox", O_RDWR);
ioctl(fd, I_PUSH, "db");
```

DESCRIPTION

The **db** STREAMS module processes the byte streams generated by the SunDials dial box. The dial box generates a stream of bytes that encode the identity of the dials and the amount by which they are turned.

Each dial sample in the byte stream consists of three bytes. The first byte identifies which dial was turned and the next two bytes return the delta in signed binary format. When bound to an application using the window system, *Virtual User Input Device* events are generated. An event from a dial is constrained to lie between 0x80 and 0x87.

A stream with **db** pushed into it can emit *firm_events* as specified by the protocol of a VUID. **db** understands the **VUIDSFORMAT** and **VUIDGFORMAT** ioctls (see reference below), as defined in **/usr/include/sundev/dbio.h** and **/usr/include/sundev/vuid_event.h**. All other **ioctl()** requests are passed downstream. **db** sets the parameters of a serial port when it is opened. No **termios**(4) **ioctl()** requests should be performed on a **db** STREAMS module, as **db** expects the device parameters to remain as it set them.

IOCTLS

VUIDSFORMAT
VUIDGFORMAT These are standard *Virtual User Input Device* ioctls. See *SunView System Programmer's Guide* for a description of their operation.

FILES

/usr/include/sundev/dbio.h
/usr/include/sundev/vuid_event.h
/usr/include/sys/ioctl.h
/usr/include/sys/stream.h
/usr/include/sys/time.h

SEE ALSO

termios(4), **dialtest**(6), **dbconfig**(8)

SunView System Programmer's Guide,
SunDials Programmers Guide

BUGS

VUIDSADDR and **VUIDGADDR** are not supported.

WARNING

The SunDials dial box must be used with a serial port.

NAME

des – DES encryption chip interface

CONFIG — SUN-3 SYSTEM

device des0 at obio ? csr 0x1c0000

CONFIG — SUN-3x SYSTEM

device des0 at obio ? csr 0x66002000

CONFIG — SUN-4 SYSTEM

device des0 at obio ? csr 0xfe000000

SYNOPSIS

#include <sys/des.h>

DESCRIPTION

The **des** driver provides a high level interface to the AmZ8068 Data Ciphering Processor, a hardware implementation of the NBS Data Encryption Standard.

The high level interface provided by this driver is hardware independent and could be shared by future drivers in other systems.

The interface allows access to two modes of the DES algorithm: Electronic Code Book (ECB) and Cipher Block Chaining (CBC). All access to the DES driver is through **ioctl**(2) calls rather than through reads and writes; all encryption is done in-place in the user's buffers.

IOCTLS

The ioctls provided are:

DESIOCBLOCK

This call encrypts/decrypts an entire buffer of data, whose address and length are passed in the '**struct desparams**' addressed by the argument. The length must be a multiple of 8 bytes.

DESIOCQUICK

This call encrypts/decrypts a small amount of data quickly. The data is limited to **DES_QUICKLEN** bytes, and must be a multiple of 8 bytes. Rather than being addresses, the data is passed directly in the '**struct desparams**' argument.

FILES

/dev/des

SEE ALSO

des(1), **des_crypt**(3)

Federal Information Processing Standards Publication 46

AmZ8068 DCP Product Description, Advanced Micro Devices

NAME

dkio – generic disk control operations

DESCRIPTION

All Sun disk drivers support a set of **ioctl**(2) requests for disk formatting and labeling operations. Basic to these **ioctl**() requests are the definitions in **/usr/include/sun/dkio.h**:

```
/*
 * Structures and definitions for disk I/O control commands
 */
/* Controller and disk identification */
struct dk_info {
        int     dki_ctlr;               /* controller address */
        short   dki_unit;               /* unit (slave) address */
        short   dki_ctype;              /* controller type */
        short   dki_flags;              /* flags */
};
/* controller types */
#define DKC_UNKNOWN     0
#define DKC_DSD5215     5
#define DKC_XY450       6
#define DKC_ACB4000     7
#define DKC_MD21        8
#define DKC_XD7053      11
#define DKC_CSS         12
#define DKC_NEC765      13      /* floppy on Sun386i */
#define DKC_INTEL82072  14
/* flags */
#define DKI_BAD144      0x01    /* use DEC std 144 bad sector fwding */
#define DKI_MAPTRK      0x02    /* controller does track mapping */
#define DKI_FMTTRK      0x04    /* formats only full track at a time */
#define DKI_FMTVOL      0x08    /* formats only full volume at a time */
/* Definition of a disk's geometry */
struct dk_geom {
        unsigned short  dkg_ncyl;       /* # of data cylinders */
        unsigned short  dkg_acyl;       /* # of alternate cylinders */
        unsigned short  dkg_bcyl;       /* cyl offset (for fixed head area) */
        unsigned short  dkg_nhead;      /* # of heads */
        unsigned short  dkg_bhead;      /* head offset (for Larks, etc.) */
        unsigned short  dkg_nsect;      /* # of sectors per track */
        unsigned short  dkg_intrlv;     /* interleave factor */
        unsigned short  dkg_gap1;       /* gap 1 size */
        unsigned short  dkg_gap2;       /* gap 2 size */
        unsigned short  dkg_apc;        /* alternates per cyl (SCSI only) */
        unsigned short  dkg_extra[9];   /* for compatible expansion */
};
/* Partition map (part of dk_label) */
struct dk_map {
        long    dkl_cylno;      /* starting cylinder */
        long    dkl_nblk;       /* number of blocks */
};
```

```
/* Floppy characteristics */
struct fdk_char {
        u_char  medium;         /* medium type (scsi floppy only) */
        int     transfer_rate;  /* transfer rate */
        int     ncyl;           /* number of cylinders */
        int     nhead;          /* number of heads */
        int     sec_size;       /* sector size */
        int     secptrack;      /* sectors per track */
        int     steps;          /* number of steps per */
};
/* Used by FDKGETCHANGE, returned state of the sense disk change bit. */
#define FDKGC_HISTORY       0x01    /* disk has changed since last call */
#define FDKGC_CURRENT       0x02    /* current state of disk change */
/* disk I/O control commands */
#define DKIOCINFO       _IOR(d, 8, struct dk_info)      /* Get info */
#define DKIOCGGEOM      _IOR(d, 2, struct dk_geom)      /* Get geometry */
#define DKIOCSGEOM      _IOW(d, 3, struct dk_geom)      /* Set geometry */
#define DKIOCGPART      _IOR(d, 4, struct dk_map)       /* Get partition info */
#define DKIOCSPART      _IOW(d, 5, struct dk_map)       /* Set partition info */
#define DKIOCWCHK       _IOWR(d, 115, int)              /* Toggle write check */
/* floppy I/O control commands */
#define FDKIOGCHAR      _IOR(d, 114, struct fdk_char)   /* Get floppy characteristics */
#define FDKEJECT        _IO(d, 112)                     /* Eject floppy */
#define FDKGETCHANGE    _IOR(d, 111, int)               /* Get disk change status */
```

The **DKIOCINFO** ioctl returns a **dk_info** structure which tells the type of the controller and attributes about how bad-block processing is done on the controller. The **DKIOCGPART** and **DKIOCSPART** get and set the controller's current notion of the partition table for the disk (without changing the partition table on the disk itself), while the **DKIOCGGEOM** and **DKIOCSGEOM** ioctls do similar things for the per-drive geometry information. The **DKIOCWCHK** enables or disables a disk's write check capabilities. The **FDKIOGCHAR** ioctl returns an **fdk_char** structure which gives the characteristics of the floppy diskette. The **FDKEJECT** ioctl ejects the floppy diskette. The **FDKGETCHANGE** returns the status of the diskette changed signal from the floppy interface.

FILES

/usr/include/sun/dkio.h

SEE ALSO

fd(4S), **ip**(4P), **sd**(4S), **xd**(4S), **xy**(4S), **dkctl**(8)

NAME

drum – paging device

CONFIG

None; included with standard system.

SYNOPSIS

#include <fcntl.h>

open("/dev/drum", *mode***);**

DESCRIPTION

This file refers to the paging device in use by the system. This may actually be a subdevice of one of the disk drivers, but in a system with paging interleaved across multiple disk drives it provides an indirect driver for the multiple drives.

FILES

/dev/drum

BUGS

Reads from the drum are not allowed across the interleaving boundaries. Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.

NAME

fb – driver for Sun console frame buffer

CONFIG

None; included in standard system.

DESCRIPTION

The **fb** driver provides indirect access to a Sun frame buffer. It is an indirect driver for the Sun workstation console's frame buffer. At boot time, the workstation's frame buffer device is determined from information from the PROM monitor and set to be the one that **fb** will indirect to. The device driver for the console's frame buffer must be configured into the kernel so that this indirect driver can access it.

The idea behind this driver is that user programs can open a known device, query its characteristics and access it in a device dependent way, depending on the type. **fb** redirects **open**(2V), **close**(2V), **ioctl**(2), and **mmap**(2) calls to the real frame buffer. All Sun frame buffers support the same general interface; see **fbio**(4S).

FILES

/dev/fb

SEE ALSO

close(2V), **ioctl**(2), **mmap**(2), **open**(2V), **fbio**(4S)

NAME

fbio – frame buffer control operations

DESCRIPTION

All Sun frame buffers support the same general interface that is defined by **<sun/fbio.h>**. Each responds to an **FBIOGTYPE ioctl(2)** request which returns information in a **fbtype** structure.

Each device has an **FBTYPE** which is used by higher-level software to determine how to perform graphics functions. Each device is used by opening it, doing an **FBIOGTYPE ioctl()** to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

Full-fledged frame buffers (that is, those that run SunView1) implement an **FBIOGPIXRECT ioctl()** request, which returns a pixrect. This call is made only from inside the kernel. The returned pixrect is used by **win**(4S) for cursor tracking and colormap loading.

FBIOSVIDEO and **FBIOGVIDEO** are general-purpose **ioctl()** reuqests for controlling possible video features of frame buffers. These **ioctl()** requests either set or return the value of a flags integer. At this point, only the **FBVIDEO_ON** option is available, controlled by **FBIOSVIDEO**. **FBIOGVIDEO** returns the current video state.

The **FBIOSATTR** and **FBIOGATTR ioctl()** requests allow access to special features of newer frame buffers. They use the **fbsattr** and **fbgattr** structures.

Some color frame buffers support the **FBIOPUTCMAP** and **FBIOGETCMAP ioctl()** reuqests, which provide access to the colormap. They use the **fbcmap** structure.

SEE ALSO

ioctl(2), **mmap**(2), **bw***(4S), **cg***(4S), **gp***(4S), **fb**(4S), **win**(4S)

BUGS

The **FBIOSATTR** and **FBIOGATTR ioctl()** requests are only supported by frame buffers which emulate older frame buffer types. For example, **cgfour**(4S) frame buffers emulate **bwtwo**(4S) frame buffers. If a frame buffer is emulating another frame buffer, **FBIOGTYPE** returns the emulated type. To get the real type, use **FBIOGATTR**.

NAME

fd – disk driver for Floppy Disk Controllers

CONFIG — Sun386i SYSTEMS

controller fdc0 at atmem ? csr 0x1000 dmachan 2 irq 6 priority 2
disk fd0 at fdc0 drive 0 flags 0

CONFIG — SUN-3/80 SYSTEMS

controller fdc0 at obio ? csr 0x6e000000 priority 6 vector fdintr 0x5c
disk fd0 at fdc0 drive 0 flags 0

CONFIG — SPARCstation 1 SYSTEMS

device-driver fd

AVAILABILITY

Sun386i, Sun-3/80, and SPARCstation 1 systems only.

DESCRIPTION

The **fd** driver provides an interface to floppy disks using the Intel 82072 disk controller on Sun386i, Sun-3/80 and SPARCstation 1 systems.

The minor device number in files that use the floppy interface encodes the unit number as well as the partition. The bits of the minor device number are defined as **rrruuppp** where **r**=reserved, **u**=unit, and **p**=partition. The unit number selects a particular floppy drive for the controller. The partition number picks one of eight partitions [**a-h**].

When the floppy is first opened the driver looks for a label in logical block 0 of the diskette. If a label is found, the geometry and partition information from the label will be used on each access thereafter. The driver first assumes high density characteristics when it tries to read the label. If the read fails it will try the read again using low density characteristics. If both attempts to read the label fail, the open will fail. Use the **FNDELAY** flag when opening an unformatted diskette as a signal to the driver that it should not attempt to access the diskette. If block 0 is read successfully, but a label is not found, the open will fail for the block interface. Using the raw interface, the open will succeed even if the diskette is unlabeled. Default geometry and partitioning are assumed if the diskette is unlabeled.

The default partitions are:

a	–> 0, N-1
b	–> N-1, N
c	–> 0, N

where N is the number of cylinders on the diskette.

The **fd** driver supports both block and raw interfaces. The block files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single **read**(2V) or **write**(2V) call usually results in one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra '**r**'.

FILES — Sun386i SYSTEMS

1.44 MB Floppy Disk Drives:

/dev/fd0a	block file
/dev/fd0c	block file
/dev/rfd0a	raw file
/dev/rfd0c	raw file

720 K Floppy Disk Drives:

/dev/fdl0a	block file
/dev/fdl0c	block file
/dev/rfdl0a	raw file
/dev/rfdl0c	raw file

FILES — SUN-3/80 and SPARCstation 1 SYSTEMS

Note: the **fd** driver on Sun-3/80 and SPARCstation 1 systems auto-senses the density of the floppy.

/dev/fd0[a-c]	block file
/dev/fd0	block file (same as **/dev/fd0c**)
/dev/rfd0[a-c]	raw file
/dev/rfd0	raw file (same as **/dev/rfd0c**)

SEE ALSO

read(2V), **write**(2V), **dkio**(4S)

DIAGNOSTICS — Sun386i SYSTEMS

fd drv %d, trk %d: %s

A command such as read or write encountered a format-related error condition. The value of *%s* is derived from the error number given by the controller, indicating the nature of the error. The track number is relative to the beginning of the partition involved.

fd drv %d, blk %d: %s

A command such as read or write encountered an error condition related to I/O. The value of *%s* is derived from the error number returned by the controller and indicates the nature of the error. The block number is relative to the start of the partition involved.

fd controller: %s

An error occurred in the controller. The value of *%s* is derived from the status returned by the controller and specifies the error encountered.

fd(%d):%s please insert

I/O was attempted while the floppy drive door was not latched. The value of *%s* indicates which disk was expected to be in the drive.

DIAGNOSTICS — SUN-3/80 and SPARCstation 1 SYSTEMS

fd%d: %s failed (%x %x %x)

The command, %s, failed after several retries on drive %d. The three hex values in parenthesis are the contents of status register 0, status register 1, and status register 2 of the Intel 82072 Floppy Disk Controller on completion of the command as documented in the data sheet for that part. This error message is usually followed by one of the following, interpreting the bits of the status register:

fd%d: not writable
fd%d: crc error
fd%d: overrun/underrun
fd%d: bad format
fd%d: timeout

NOTES

Floppy diskettes have 18 sectors per track, and can cross a track (though not a cylinder) boundary without lossing data, so when using **dd**(1) to or from a diskette, you should specify **bs=18k** or multiples thereof.

NAME

filio – ioctls that operate directly on files, file descriptors, and sockets

SYNOPSIS

#include <sys/filio.h>

DESCRIPTION

The IOCTL's listed in this manual page apply directly to files, file descriptors, and sockets, independent of any underlying device or protocol.

Note: the **fcntl**(2V) system call is the primary method for operating on file descriptors as such, rather than on the underlying files.

IOCTLS for File Descriptors

FIOCLEX The argument is ignored. Set the close-on-exec flag for the file descriptor passed to **ioctl**. This flag is also manipulated by the **F_SETFD** command of **fcntl**(2V).

FIONCLEX The argument is ignored. Clear the close-on-exec flag for the file descriptor passed to **ioctl**.

IOCTLs for Files

FIONREAD The argument is a pointer to a **long**. Set the value of that **long** to the number of immediately readable characters from whatever the descriptor passed to **ioctl** refers to. This works for files, pipes, sockets, and terminals.

FIONBIO The argument is a pointer to an **int**. Set or clear non-blocking I/O. If the value of that **int** is a 1 (one) the descriptor is set for non-blocking I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for non-blocking I/O.

FIOASYNC The argument is a pointer to an **int**. Set or clear asynchronous I/O. If the value of that **int** is a 1 (one) the descriptor is set for asynchronous I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for asynchronous I/O.

FIOSETOWN The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl** to the value of that **int**.

FIOGETOWN The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl**.

SEE ALSO

ioctl(2), **fcntl**(2V), **getsockopt**(2), **sockio**(4)

NAME

fpa – Sun-3/Sun-3x floating-point accelerator

CONFIG — SUN-3/SUN-3X SYSTEMS

device fpa0 at virtual ? csr 0xe0000000

SYNOPSIS

#include <sundev/fpareg.h>
open("/dev/fpa", flags);

DESCRIPTION

FPA and FPA+ are compatible floating point accelerators available on certain Sun-3 and Sun-3x systems. They provide hardware contexts for simultaneous use by up to 32 processes. The same **fpa** device driver manages either FPA or FPA+ hardware.

Processes access the device using **open**(2V) and **close**(2V) system calls, and the FPA is automatically mapped into the process' address space by SunOS. This is normally provided transparently at compile time by a compiler option, such as the **–ffpa** option to **cc**(1V).

The valid **ioctl**(2) system calls are used only by diagnostics and by system administration programs, such as **fpa_download**(8).

IOCTLS

FPA_ACCESS_OFF	Clear **FPA_ACCESS_BIT** in FPA state register to disable access to constants RAM using FPA load pointer.
FPA_ACCESS_ON	Set **FPA_ACCESS_BIT** in FPA state register to enable access to constants RAM using FPA load pointer.
FPA_FAIL	Disable the FPA.
FPA_GET_DATAREGS	Return the contents of 8 FPA registers.
FPA_INIT_DONE	Called when downloading is complete. Allows multiple users to access the FPA.
FPA_LOAD_OFF	Set **FPA_LOAD_BIT** in FPA state register to disable access to microstore or map RAM via FPA load pointer.
FPA_LOAD_ON	Set **FPA_LOAD_BIT** in FPA state register to enable access to microstore or map RAM using FPA load pointer.

The following two **ioctl()** requests are for diagnostic use only. **fpa** must be compiled with **FPA_DIAGNOSTICS_ONLY** defined to enable these two calls.

FPA_WRITE_STATE	Overwrite the FPA state register.
FPA_WRITE_HCP	Write to the hard clear pipe register.

ERRORS

The following error messages are returned by **open** system calls only.

EBUSY	All 32 FPA contexts are being used.
EEXIST	The current process has already opened **/dev/fpa**.
EIO	Downloading has not completed, so only 1 root process can have the FPA open at a time.
ENETDOWN	FPA is disabled.
ENOENT	68881 chip does not exist.
ENXIO	FPA board does not exist.

The following error messages are returned by **ioctl** system calls only.

EINVAL	Invalid ioctl. This may occur if diagnostic only ioctls, **FPA_WRITE_STATE** or **FPA_WRITE_HCP**, are used with a driver which didn't compile in those calls.

EPERM	All ioctl calls except for **FPA_GET_DATAREGS** require root execution level.
EPIPE	The FPA pipe is not clear.

FILES

/dev/fpa device file for both FPA and FPA+.

SEE ALSO

cc(1V), **close**(2V), **ioctl**(2), **open**(2V) **fpa_download**(8), **fparel**(8), **fpaversion**(8)

DIAGNOSTICS

If hardware problems are detected then all processes with **/dev/fpa** open are killed, and future opens of **/dev/fpa** are disabled.

NAME

gpone – graphics processor

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

device gpone0 at vme24d16 ? csr 0x210000 **# GP or GP+**
device gpone0 at vme24d32 ? csr 0x240000 **# GP2**

DESCRIPTION

The **gpone** interface provides access to the optional Graphics Processor Board (GP).

The hardware consumes 64 kilobytes of VME bus address space. The GP board starts at standard address 0x210000 and must be configured for interrupt level 4.

IOCTLS

The graphics processor responds to a number of ioctl calls as described here. One of the calls uses a **gp1fbinfo** structure that looks like this:

```
struct  gp1fbinfo {
        int         fb_vmeaddr;    /* physical color board address */
        int         fb_hwwidth;    /* fb board width */
        int         fb_hwheight;   /* fb board height */
        int         addrdelta;     /* phys addr diff between fb and gp */
        caddr_t     fb_ropaddr;    /* cg2 va thru kernelmap */
        int         fbunit;        /* fb unit to use  for a,b,c,d */
};
```

The ioctl call looks like this:

```
ioctl(file, request, argp)
int file, request;
```

argp is defined differently for each GP ioctl request and is specified in the descriptions below.

The following ioctl commands provide for transferring data between the graphics processor and color boards and processes.

GP1IO_PUT_INFO
: Passes information about the frame buffer into driver. **argp** points to a **struct gp1fbinfo** which is passed to the driver.

GP1IO_GET_STATIC_BLOCK
: Hands out a static block from the GP. **argp** points to an **int** which is returned from the driver.

GP1IO_FREE_STATIC_BLOCK
: Frees a static block from the GP. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_GBUFFER_STATE
: Checks to see if there is a buffer present on the GP. **argp** points to an **int** which is returned from the driver.

GP1IO_CHK_GP
: Restarts the GP if necessary. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_RESTART_COUNT
: Returns the number of restarts of a GP since power on. Needed to differentiate SIGXCPU calls in user processes. **argp** points to an **int** which is returned from the driver.

GP1IO_REDIRECT_DEVFB
: Configures **/dev/fb** to talk to a graphics processor device. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_REQDEV
: Returns the requested minor device. **argp** points to a **dev_t** which is returned from the driver.

GP1IO_GET_TRUMINORDEV

Returns the true minor device. **argp** points to a **char** which is returned from the driver.

The graphics processor driver also responds to the **FBIOGTYPE**, ioctl which a program can use to inquire as to the characteristics of the display device, the **FBIOGINFO**, ioctl for passing generic information, and the **FBIOGPIXRECT** ioctl so that SunWindows can run on it. See **fbio**(4S).

FILES

/dev/fb
/dev/gpone[0-3][abcd]

SEE ALSO

fbio(4S), **mmap**(2), **gpconfig**(8)

SunCGI Reference Manual

DIAGNOSTICS

The Graphics Processor has been restarted. You may see display garbage as a result.

NAME

icmp – Internet Control Message Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed through a "raw socket" for network monitoring and diagnostic functions. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from **getprotobyname** (see **getprotoent**(3N)). ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the **connect**(2) call may also be used to fix the destination for future packets (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) system calls may be used).

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the holder of a raw socket with the IP header and options intact.

ICMP is an unreliable datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP "redirect" message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided using the ICMP raw socket.

ERRORS

A socket operation may fail with one of the following errors returned:

EISCONN	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
ENOTCONN	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
ENOBUFS	when the system runs out of memory for an internal data structure;
EADDRNOTAVAIL	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

connect(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P), **routing**(4N)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1064-01)

BUGS

Replies to ICMP "echo" messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

ie – Intel 10 Mb/s Ethernet interface

CONFIG — SUN-4 SYSTEM

device ie0 at obio ? csr 0x6000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d16 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3x SYSTEM

device ie0 at obio ? csr 0x65000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d32 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d32 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3 SYSTEM

device ie0 at obio ? csr 0xc0000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d32 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d32 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3E SYSTEM

device ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74

CONFIG — SUN386i SYSTEM

device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network through a controller using the Intel 82586 LAN Coprocessor chip. For a general description of network interfaces see **if**(4N).

ie0 specifies a CPU-board-resident interface, except on a Sun-3E where **ie0** is the Sun-3/E Ethernet expansion board. **ie1** specifies a Multibus Intel Ethernet interface for use with a VME adapter. **ie2** and **ie3** specify SunNet Ethernet/VME Controllers, also known as a Sun-3/E Ethernet expansion boards.

SEE ALSO

if(4N), **le**(4S)

DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

ie%d: Ethernet jammed

Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

ie%d: no carrier

The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

ie%d: lost interrupt: resetting

The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

ie%d: iebark reset

The 82586 failed to complete a watchdog timeout command in the allotted time. The driver recovers by resetting itself and the chip.

ie%d: WARNING: requeuing

The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

ie%d: panic: scb overwritten

The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

ie%d: giant packet

Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of this message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

NAME

if – general properties of network interfaces

DESCRIPTION

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, **lo**(4), do not.

At boot time, each interface with underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages can be routed through it. Most interfaces require some part of their address specified with an **SIOCSIFADDR** IOCTL before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets using **arp**(4P)), the entire address specified in the ioctl is used.

The following **ioctl** calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an **ifreq** structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16];           /* name of interface (e.g. "ec0") */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                short   ifru_flags;
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_flags       ifr_ifru.ifru_flags     /* flags */
};
```

SIOCSIFADDR Set interface address. Following the address assignment, the ''initialization'' routine for the interface is called.

SIOCGIFADDR Get interface address.

SIOCSIFDSTADDR Set point to point address for interface.

SIOCGIFDSTADDR Get point to point address for interface.

SIOCSIFFLAGS Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

SIOCGIFFLAGS Get interface flags.

SIOCGIFCONF Get interface configuration list. This request takes an **ifconf** structure (see below) as a value-result parameter. The **ifc_len** field should be initially set to the size of the buffer pointed to by **ifc_buf**. On return it will contain the length, in bytes, of the configuration list.

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct	ifconf {
	int	ifc_len;		/* size of associated buffer */
	union {
		caddr_t ifcu_buf;
		struct	ifreq *ifcu_req;
	} ifc_ifcu;
#define ifc_buf	ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req	ifc_ifcu.ifcu_req /* array of structures returned */
};
```

SIOCADDMULTI Enable a multicast address for the interface. A maximum of 64 multicast addresses may be enabled for any given interface.

SIOCDELMULTI Disable a previously set multicast address.

SIOCSPROMISC Toggle promiscuous mode.

SEE ALSO

arp(4P), **lo**(4)

NAME

inet – Internet protocol family

SYNOPSIS

options INET

#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family provides protocol support for the **SOCK_STREAM**, **SOCK_DGRAM**, and **SOCK_RAW** socket types.

PROTOCOLS

The Internet protocol family is comprised of the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP is used to support the **SOCK_STREAM** abstraction while UDP is used to support the **SOCK_DGRAM** abstraction; see **tcp**(4P) and **udp**(4P). A raw interface to IP is available by creating an Internet socket of type **SOCK_RAW;** see **ip**(4P). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see **icmp**(4P). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see **arp**(4P).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; the most-significant bit is zero in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of local networks may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following **ioctl**(2) commands on a datagram socket in the Internet domain; they have the same form as the **SIOCIFADDR** command (see **intro**(4)).

SIOCSIFNETMASK Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

SIOCGIFNETMASK Get interface network mask.

ADDRESSING

IP addresses are four byte quantities, stored in network byte order (on Sun386i systems these are word and byte reversed).

Sockets in the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; see **intro**(3).

The **sin_addr** field of the **sockaddr_in** structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value **INADDR_ANY** may be used in this field to effect "wildcard" matching. Given in a **bind**(2) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The **sockaddr_in** structure given in the **bind**(2) call must specify an **in_addr** value of either **IPADDR_ANY** or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAIL. When a **connect**(2) call is made for a socket that has a wildcard local address, the system sets the **sin_addr** field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The **sin_port** field of the **sockaddr_in** structure specifies a port number used by TCP or UDP. The local port address specified in a **bind**(2) call is restricted to be greater than **IPPORT_RESERVED** (defined in **<netinet/in.h>**) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than **IPPORT_RESERVED**. A unique local port address is also picked when a socket which is not bound is used in a **connect**(2) or **send**(2) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling **socket**(2) and then **connect**(2), and to send UDP datagrams with a **socket**(2) call followed by a **send**(2) call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the **SO_REUSEADDR** socket option with **setsockopt** (see **getsockopt**(2)).

SEE ALSO

bind(2), **connect**(2), **getsockopt**(2), **ioctl**(2), **send**(2), **socket**(2), **intro**(3), **byteorder**(3N), **gethostent**(3N), **getnetent**(3N), **getprotoent**(3N), **getservent**(3N), **inet**(3N), **intro**(4), **arp**(4P), **icmp**(4P), **ip**(4P) **tcp**(4P), **udp**(4P)

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985.
A 4.2BSD Interprocess Communication Primer

WARNING

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

ip – Internet Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly using a ''raw socket.'' See **tcp**(4P) and **udp**(4P). The protocol options defined in the IP specification may be set in outgoing datagrams.

Raw IP sockets are connectionless and are normally used with the **sendto** and **recvfrom** calls, (see **send**(2) and **recv**(2)) although the **connect**(2) call may also be used to fix the destination for future datagrams (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) calls may be used). If **proto** is zero, the default protocol, **IPPROTO_RAW**, is used. If **proto** is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; Received datagrams are returned with the IP header and options intact.

A single socket option, **IP_OPTIONS**, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with **setsockopt** (see **getsockopt**(2)). The **getsockopt**(2) call returns the IP options set in the last **setsockopt** call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in **setsockopt** matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option **SO_DONTROUTE** may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram via, and possibly an intermediate gateway, based on an entry in the routing table. See **routing**(4N). When **SO_DONTROUTE** is set, the datagram will be sent via the interface whose network number or full IP address matches the destination address. If no interface matches, the error ENETUNRCH will be returned.

Datagrams flow through the IP layer in two directions: from the network **ip** to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See **icmp**(4P).

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable **ipcksum** to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A "time exceeded" ICMP message will be sent if the "time to live" field in the IP header drops to zero in the process of forwarding a datagram. A "destination unreachable" message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP "source quench" message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

ERRORS

A socket operation may fail with one of the following errors returned:

EACCESS	when specifying an IP broadcast destination address if the caller is not the super-user;
EISCONN	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
EMSGSIZE	when sending datagram that is too large for an interface, but is not allowed be fragmented (such as broadcasts);
ENETUNREACH	when trying to establish a connection or send a datagram, if there is no matching entry in the routing table, or if an ICMP "destination unreachable" message is received.
ENOTCONN	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
ENOBUFS	when the system runs out of memory for fragmentation buffers or other internal data structure;
EADDRNOTAVAIL	when an attempt is made to create a socket with a local address that matches no network interface, or when specifying an IP broadcast destination address and the network interface does not support broadcast;

The following errors may occur when setting or getting IP options:

EINVAL	An unknown socket option name was given.
EINVAL	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO

connect(2), **getsockopt**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **icmp**(4P), **inet**(4F) **routing**(4N), **tcp**(4P), **udp**(4P)

Postel, Jon, "*Internet Protocol - DARPA Internet Program Protocol Specification*," RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1063-01)

BUGS

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

kb – Sun keyboard STREAMS module

CONFIG

pseudo-device kb*number*

SYNOPSIS

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sundev/vuid_event.h>
#include <sundev/kbio.h>
#include <sundev/kbd.h>

ioctl(fd, I_PUSH, "kb");

DESCRIPTION

The **kb** STREAMS module processes byte streams generated by Sun keyboards attached to a CPU serial or parallel port. Definitions for altering keyboard translation, and reading events from the keyboard, are in **<sundev/kbio.h>** and **<sundev/kbd.h>**. *number* specifies the maximum number of keyboards supported by the system.

kb recognizes which keys have been typed using a set of tables for each known type of keyboard. Each translation table is an array of 128 16-bit words (**unsigned shorts**). If an entry in the table is less than 0x100, it is treated as an ISO 8859/1 character. Higher values indicate special characters that invoke more complicated actions.

Keyboard Translation Mode

The keyboard can be in one of the following translation modes:

TR_NONE	Keyboard translation is turned off and up/down key codes are reported.
TR_ASCII	ISO 8859/1 codes are reported.
TR_EVENT	**firm_events** are reported (see *SunView Programmer's Guide*).
TR_UNTRANS_EVENT	**firm_events** containing unencoded keystation codes are reported for all input events within the window system.

Keyboard Translation-Table Entries

All instances of the **kb** module share seven translation tables used to convert raw keystation codes to event values. The tables are:

Unshifted	Used when a key is depressed and no shifts are in effect.
Shifted	Used when a key is depressed and a Shift key is being held down.
Caps Lock	Used when a key is depressed and Caps Lock is in effect.
Alt Graph	Used when a key is depressed and the Alt Graph key is being held down.
Num Lock	Used when a key is depressed and Num Lock is in effect.
Controlled	Used when a key is depressed and the Control key is being held down (regardless of whether a Shift key or the Alt Graph is being held down, or whether Caps Lock or Num Lock is in effect).
Key Up	Used when a key is released.

Each key on the keyboard has a "key station" code which is a number from 0 to 127. This number is used as an index into the translation table that is currently in effect. If the corresponding entry in that translation table is a value from 0 to 255, this value is treated as an ISO 8859/1 character, and that character is the result of the translation.

If the entry is a value above 255, it is a "special" entry. Special entry values are classified according to the value of the high-order bits. The high-order value for each class is defined as a constant, as shown in the list below. The value of the low-order bits, when added to this constant, distinguishes between keys within each class:

SHIFTKEYS 0x100
: A shift key. The value of the particular shift key is added to determine which shift mask to apply:

CAPSLOCK 0	"Caps Lock" key.
SHIFTLOCK 1	"Shift Lock" key.
LEFTSHIFT 2	Left-hand "Shift" key.
RIGHTSHIFT 3	Right-hand "Shift" key.
LEFTCTRL 4	Left-hand (or only) "Control" key.
RIGHTCTRL 5	Right-hand "Control" key.
ALTGRAPH 9	"Alt Graph" key.
ALT 10	"Alternate" key on the Sun-3 keyboard, or "Alt" key on the Sun-4 keyboard.
NUMLOCK 11	"Num Lock" key.

BUCKYBITS 0x200
: Used to toggle mode-key-up/down status without altering the value of an accompanying ISO 8859/1 character. The actual bit-position value, minus 7, is added.

METABIT 0	The "Meta" key was pressed along with the key. This is the only user-accessible bucky bit. It is ORed in as the 0x80 bit; since this bit is a legitimate bit in a character, the only way to distinguish between, for example, 0xA0 as META+0x20 and 0xA0 as an 8-bit character is to watch for "META key up" and "META key down" events and keep track of whether the META key was down.
SYSTEMBIT 1	The "System" key was pressed. This is a place holder to indicate which key is the system-abort key.

FUNNY 0x300
: Performs various functions depending on the value of the low 4 bits:

NOP 0x300	Does nothing.
OOPS 0x301	Exists, but is undefined.
HOLE 0x302	There is no key in this position on the keyboard, and the position-code should not be used.
NOSCROLL 0x303	Alternately sends CTRL-S and CTRL-Q characters.
CTRLS 0x304	Sends CTRL-S character and toggles NOSCROLL key.
CTRLQ 0x305	Sends CTRL-Q character and toggles NOSCROLL key.
RESET 0x306	Keyboard reset.
ERROR 0x307	The keyboard driver detected an internal error.
IDLE 0x308	The keyboard is idle (no keys down).
COMPOSE 0x309	This key is the COMPOSE key; the next two keys should comprise a two-character "COMPOSE key" sequence.

	NONL 0x30A	Used only in the Num Lock table; indicates that this key is not affected by the Num Lock state, so that the translation table to use to translate this key should be the one that would have been used had Num Lock not been in effect.
	0x30B — 0x30F	Reserved for nonparameterized functions.

FA_CLASS 0x400

This key is a "floating accent" or "dead" key. When this key is pressed, the next key generates an event for an accented character; for example, "floating accent grave" followed by the "a" key generates an event with the ISO 8859/1 code for the "a with grave accent" character. The low-order bits indicate which accent; the codes for the individual "floating accents" are as follows:

FA_UMLAUT 0x400	umlaut
FA_CFLEX 0x401	circumflex
FA_TILDE 0x402	tilde
FA_CEDILLA 0x403	cedilla
FA_ACUTE 0x404	acute accent
FA_GRAVE 0x405	grave accent

STRING 0x500

The low-order bits index a table of strings. When a key with a **STRING** entry is depressed, the characters in the null-terminated string for that key are sent, character by character. The maximum length is defined as:

KTAB_STRLEN 10

Individual string numbers are defined as:

HOMEARROW	0x00
UPARROW	0x01
DOWNARROW	0x02
LEFTARROW	0x03
RIGHTARROW	0x04

String numbers 0x05 — 0x0F are available for custom entries.

FUNCKEYS 0x600

Function keys. The next-to-lowest 4 bits indicate the group of function keys:

LEFTFUNC 0x600
RIGHTFUNC 0x610
TOPFUNC 0x620
BOTTOMFUNC 0x630

The low 4 bits indicate the function key number within the group:

LF(*n*)	(LEFTFUNC+(*n*)-1)
RF(*n*)	(RIGHTFUNC+(*n*)-1)
TF(*n*)	(TOPFUNC+(*n*)-1)
BF(*n*)	(BOTTOMFUNC+(*n*)-1)

There are 64 keys reserved for function keys. The actual positions may not be on left/right/top/bottom of the keyboard, although they usually are.

PADKEYS 0x700

This key is a "numeric keypad key." These entries should appear only in the Num Lock translation table; when Num Lock is in effect, these events will be generated by pressing keys on the right-hand keypad. The low-order bits indicate which key; the codes for the individual keys are as follows:

PADEQUAL 0x700	"=" key
PADSLASH 0x701	"/" key
PADSTAR 0x702	"*" key
PADMINUS 0x703	"-" key
PADSEP 0x704	"," key
PAD7 0x705	"7" key
PAD8 0x706	"8" key
PAD9 0x707	"9" key
PADPLUS 0x708	"+" key
PAD4 0x709	"4" key
PAD5 0x70A	"5" key
PAD6 0x70B	"6" key
PAD1 0x70C	"1" key
PAD2 0x70D	"2" key
PAD3 0x70E	"3" key
PAD0 0x70F	"0" key
PADDOT 0x710	"." key
PADENTER 0x711	"Enter" key

In **TR_ASCII** mode, when a function key is pressed, the following escape sequence is sent:

ESC[0 . . . . 9z

where ESC is a single escape character and "0 . .. 9" indicates the decimal representation of the function-key value. For example, function key **R1** sends the sequence:

ESC[208z

because the decimal value of RF(1) is 208. In **TR_EVENT** mode, if there is a VUID event code for the function key in question, an event with that event code is generated; otherwise, individual events for the characters of the escape sequence are generated.

Keyboard Compatibility Mode

kb is in "compatibility mode" when it starts up. In this mode, when the keyboard is in the **TR_EVENT** translation mode, ISO 8859/1 characters from the "upper half" of the character set (that is, characters with the 8th bit set) are presented as events with codes in the **ISO_FIRST** range (as defined in **<sundev/vuid_event.h>**). The event code is **ISO_FIRST** plus the character value. This is for backwards compatibility with older versions of the keyboard driver. If compatibility mode is turned off, ISO 8859/1 characters are presented as events with codes equal to the character code.

IOCTLS

The following **ioctl( )** requests set and retrieve the current translation mode of a keyboard:

KIOCTRANS The argument is a pointer to an **int**. The translation mode is set to the value in the **int** pointed to by the argument.

KIOCGTRANS The argument is a pointer to an **int**. The current translation mode is stored in the **int** pointed to by the argument.

ioctl( ) requests for changing and retrieving entries from the keyboard translation table use the **kiockeymap** structure:

```
struct  kiockeymap {
        int      kio_tablemask; /* Translation table (one of: 0, CAPSMASK,
                                   SHIFTMASK, CTRLMASK, UPMASK,
                                   ALTGRAPHMASK, NUMLOCKMASK) */
#define KIOCABORT1   -1         /* Special "mask": abort1 keystation */
#define KIOCABORT2   -2         /* Special "mask": abort2 keystation */
        u_char  kio_station;    /* Physical keyboard key station (0-127) */
        u_short kio_entry;      /* Translation table station's entry */
        char    kio_string[10]; /* Value for STRING entries (null terminated) */
};
```

KIOCSKEY The argument is a pointer to a **kiockeymap** structure. The translation table entry referred to by the values in that structure is changed.

kio_tablemask specifies which of the five translation tables contains the entry to be modified:

UPMASK 0x0080	"Key Up" translation table.
NUMLOCKMASK 0x0800	"Num Lock" translation table.
CTRLMASK 0x0030	"Controlled" translation table.
ALTGRAPHMASK 0x0200	"Alt Graph" translation table.
SHIFTMASK 0x000E	"Shifted" translation table.
CAPSMASK 0x0001	"Caps Lock" translation table.
(No shift keys pressed or locked)	"Unshifted" translation table.

kio_station specifies the keystation code for the entry to be modified. The value of **kio_entry** is stored in the entry in question. If **kio_entry** is between STRING and STRING+15, the string contained in **kio_string** is copied to the appropriate string table entry. This call may return **EINVAL** if there are invalid arguments.

There are a couple special values of **kio_tablemask** that affect the two step "break to the PROM monitor" sequence. The usual sequence is **SETUP–a** or **L1–a**. If **kio_tablemask** is **KIOCABORT1** then the value of **kio_station** is set to be the first keystation in the sequence. If **kio_tablemask** is **KIOCABORT2** then the value of **kio_station** is set to be the second keystation in the sequence.

KIOCGKEY The argument is a pointer to a **kiockeymap** structure. The current value of the keyboard translation table entry specified by **kio_tablemask** and **kio_station** is stored in the structure pointed to by the argument. This call may return **EINVAL** if there are invalid arguments.

KIOCTYPE The argument is a pointer to an **int**. A code indicating the type of the keyboard is stored in the **int** pointed to by the argument:

KB_KLUNK	Micro Switch 103SD32-2
KB_VT100	Keytronics VT100 compatible
KB_SUN2	Sun-2 keyboard
KB_SUN3	Sun-3 keyboard
KB_SUN4	Sun-4 keyboard
KB_ASCII	ASCII terminal masquerading as keyboard

–1 is stored in the **int** pointed to by the argument if the keyboard type is unknown.

KIOCLAYOUT The argument is a pointer to an **int**. On a Sun-4 keyboard, the layout code specified by the keyboard's DIP switches is stored in the **int** pointed to by the argument.

KIOCCMD The argument is a pointer to an **int**. The command specified by the value of the **int** pointed to by the argument is sent to the keyboard. The commands that can be sent are:

Commands to the Sun-2, Sun-3, and Sun-4 keyboard:

KBD_CMD_RESET	Reset keyboard as if power-up.
KBD_CMD_BELL	Turn on the bell.
KBD_CMD_NOBELL	Turn off the bell

Commands to the Sun-3 and Sun-4 keyboard:

KBD_CMD_CLICK	Turn on the click annunciator.
KBD_CMD_NOCLICK	Turn off the click annunciator.

Inappropriate commands for particular keyboard types are ignored. Since there is no reliable way to get the state of the bell or click (because we cannot query the keyboard, and also because a process could do writes to the appropriate serial driver — thus going around this **ioctl**() request) we do not provide an equivalent **ioctl**() to query its state.

KIOCSLED The argument is a pointer to an **char**. On the Sun-4 keyboard, the LEDs are set to the value specified in that **char**. The values for the four LEDs are:

LED_CAPS_LOCK	"Caps Lock" light.
LED_COMPOSE	"Compose" light.
LED_SCROLL_LOCK	"Scroll Lock" light.
LED_NUM_LOCK	"Num Lock" light.

KIOCGLED The argument is a pointer to a **char**. The current state of the LEDs is stored in the **char** pointed to by the argument.

KIOCSCOMPAT The argument is a pointer to an **int**. "Compatibility mode" is turned on if the **int** has a value of 1, and is turned off if the **int** has a value of 0.

KIOCGCOMPAT The argument is a pointer to an **int**. The current state of "compatibility mode" is stored in the **int** pointed to by the argument.

KIOCGDIRECT These **ioctl**( ) requests are supported for compatibility with the system keyboard device **/dev/kbd**. **KIOCSDIRECT** has no effect, and **KIOCGDIRECT** always returns 1.

SEE ALSO

click(1), **loadkeys**(1), **kbd**(4S), **termio**(4), **win**(4S), **keytables**(5)

SunView Programmer's Guide (describes **firm_event** format)

NAME

kbd – Sun keyboard

CONFIG

None; included in standard system.

DESCRIPTION

The **kbd** device provides access to the Sun Workstation keyboard. When opened, it provides access to the standard keyboard device for the workstation (attached either to a CPU serial or parallel port). It is a multiplexing driver; a stream referring to the standard keyboard device, with the **kb**(4M) and **ttcompat**(4M) STREAMS modules pushed on top of that device, is linked below it. Normally, this device passes input to the "workstation console" driver, which is linked above a special minor device of **kbd**, so that keystrokes appear as input on **/dev/console**; the **KIOCSDIRECT ioctl** must be used to direct input towards or away from the **/dev/kbd** device.

IOCTLS

KIOCSDIRECT The argument is a pointer to an **int**. If the value in the **int** pointed to by the argument is 1, subsequent keystrokes typed on the system keyboard will sent to **/dev/kbd**; if it is 0, subsequent keystrokes will be sent to the "workstation console" device. When the last process that has **/dev/kbd** open closes it, if keystrokes had been sent to **/dev/kbd** they are redirected back to the "workstation console" device.

KIOCGDIRECT The argument is a pointer to an **int**. If keystrokes are currently being sent to **/dev/kbd**, 1 is stored in the **int** pointed to by the argument; if keystrokes are currently being sent to the "workstation console" device, 0 is stored there.

FILES

/dev/kbd

SEE ALSO

console(4S), **kb**(4M), **ttcompat**(4M), **win**(4S), **zs**(4S)

NAME

ldterm – standard terminal STREAMS module

CONFIG

None; included by default.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>

ioctl(fd, I_PUSH, "ldterm");
```

DESCRIPTION

ldterm is a STREAMS module that provides most of the **termio**(4) terminal interface. This module does not perform the low-level device control functions specified by flags in the **c_cflag** word of the **termios** structure or by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure; those functions must be performed by the driver or by modules pushed below the **ldterm** module. All other **termio** functions are performed by **ldterm**; some of them, however, require the cooperation of the driver or modules pushed below **ldterm**, and may not be performed in some cases. These include the **IXOFF** flag in the **c_iflag** word and the delays specified in the **c_oflag** word.

Read-side Behavior

Various types of STREAMS messages are processed as follows:

M_BREAK When this message is received, either an interrupt signal is generated, or the message is treated as if it were an **M_DATA** message containing a single ASCII NUL character, depending on the state of the **BRKINT** flag.

M_DATA These messages are normally processed using the standard **termio** input processing. If the **ICANON** flag is set, a single input record ("line") is accumulated in an internal buffer, and sent upstream when a line-terminating character is received. If the **ICANON** flag is not set, other input processing is performed and the processed data is passed upstream.

If output is to be stopped or started as a result of the arrival of characters, **M_STOP** and **M_START** messages are sent downstream, respectively. If the **IXOFF** flag is set, and input is to be stopped or started as a result of flow-control considerations, **M_STOPI** and **M_STARTI** messages are sent downstream, respectively.

M_DATA messages are sent downstream, as necessary, to perform echoing.

If a signal is to be generated, a **M_FLUSH** message with a flag byte of **FLUSHR** is placed on the read queue, and if the signal is also to flush output a **M_FLUSH** message with a flag byte of **FLUSHW** is sent downstream.

M_CTL If the first byte of the message is **MC_NOCANON**, the input processing normally performed on **M_DATA** messages is disabled, and those messages are passed upstream unmodified; this is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in **TIOCREMOTE** mode connected to a program that performs this processing. If the first byte of the message is **MC_DOCANON**, the input processing is enabled. Otherwise, the message is ignored; in any case, the message is passed upstream.

M_FLUSH The read queue of the module is flushed of all its data messages, and all data in the record being accumulated is also flushed. The message is passed upstream.

M_HANGUP Data is flushed as it is for a **M_FLUSH** message, and **M_FLUSH** messages with a flag byte of **FLUSHRW** are sent upstream and downstream. Then an **M_PCSIG** message is sent upstream with a signal of **SIGCONT**, followed by the **M_HANGUP** message.

M_IOCACK The data contained within the message, which is to be returned to the process, is augmented if necessary, and the message is passed upstream.

All other messages are passed upstream unchanged.

Write-side behavior

Various types of STREAMS messages are processed as follows:

M_FLUSH The write queue of the module is flushed of all its data messages, and the message is passed downstream.

M_IOCTL The function to be performed for this **ioctl()** request by the **ldterm** module is performed, and the message is passed downstream in most cases. The **TCFLSH** and **TCXONC** **ioctl()** requests can be performed entirely in this module, so the reply is sent upstream and the message is not passed downstream.

M_DATA If the **OPOST** flag is set, or both the **XCASE** and **ICANON** flags are set, output processing is performed and the processed message is passed downstream, along with any **M_DELAY** messages generated. Otherwise, the message is passed downstream without change.

All other messages are passed downstream unchanged.

IOCTLS

The following **ioctl()** requests are processed by the **ldterm** module. All others are passed downstream.

TCGETS
TCGETA The message is passed downstream; if an acknowledgment is seen, the data provided by the driver and modules downstream is augmented and the acknowledgement is passed upstream.

TCSETS
TCSETSW
TCSETSF
TCSETA
TCSETAW
TCSETAF The parameters that control the behavior of the **ldterm** module are changed. If a mode change requires options at the stream head to be changed, a **M_SETOPT** message is sent upstream. If the **ICANON** flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If it is turned on, the **vmin** and **vtime** values at the stream head are set to 1 and 0, respectively; if it is turned on, they are set to the values specified by the **ioctl()** request. The **vmin** and **vtime** values are also set if **ICANON** is off and the values are changed by the **ioctl()** request. If the **TOSTOP** flag is turned on or off, the **tostop** mode at the stream head is turned on or off, respectively.

TCFLSH If the argument is 0, an **M_FLUSH** message with a flag byte of **FLUSHR** is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and a **M_FLUSH** message with a flag byte of **FLUSHW** is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and a **M_FLUSH** message with a flag byte of **FLUSHRW** is sent downstream and placed on the read queue.

TCXONC If the argument is 0, and output is not already stopped, an **M_STOP** message is sent downstream. If the argument is 1, and output is stopped, an **M_START** message is sent downstream. If the argument is 2, and input is not already stopped, an **M_STOPI** message is sent downstream. If the argument is 3, and input is stopped, an **M_STARTI** message is sent downstream.

SEE ALSO

console(4S), **mcp**(4S), **mti**(4S), **pty**(4), **termio**(4), **ttcompat**(4M), **zs**(4S)

NAME

ie – Intel 10 Mb/s Ethernet interface

CONFIG — SUN-4 SYSTEM

device ie0 at obio ? csr 0xf6000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d16 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3x SYSTEM

device ie0 at obio ? csr 0x65000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

CONFIG — SUN-3 SYSTEM

device ie0 at obio ? csr 0xc0000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d32 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d32 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3E SYSTEM

device ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74

CONFIG — SUN386i SYSTEM

device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network through a controller using the Intel 82586 LAN Coprocessor chip. For a general description of network interfaces see **if**(4N).

ie0 specifies a CPU-board-resident interface, except on a Sun-3E where **ie0** is the Sun-3/E Ethernet expansion board. **ie1** specifies a Multibus Intel Ethernet interface for use with a VME adapter. **ie2** and **ie3** specify SunNet Ethernet/VME Controllers, also known as a Sun-3/E Ethernet expansion boards.

SEE ALSO

if(4N), **le**(4S)

DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

ie%d: Ethernet jammed
Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

ie%d: no carrier
The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

ie%d: lost interrupt: resetting
The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

ie%d: iebark reset
The 82586 failed to complete a watchdog timeout command in the allotted time. The driver recovers by resetting itself and the chip.

ie%d: WARNING: requeuing

The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

ie%d: panic: scb overwritten

The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

ie%d: giant packet

Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of this message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

NAME

lo – software loopback network interface

SYNOPSIS

pseudo-device loop

DESCRIPTION

The **loop** device is a software loopback network interface; see **if**(4N) for a general description of network interfaces.

The **loop** interface is used for performance analysis and software testing, and to provide guaranteed access to Internet protocols on machines with no local network interfaces. A typical application is the **comsat**(8C) server which accepts notification of mail delivery through a particular port on the loopback interface.

By default, the loopback interface is accessible at Internet address 127.0.0.1 (non-standard); this address may be changed with the **SIOCSIFADDR** ioctl.

SEE ALSO

if(4N), **inet**(4F), **comsat**(8C)

DIAGNOSTICS

lo%d: can't handle af%d

The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

BUGS

It should handle all address and protocol families. An approved network address should be reserved for this interface.

NAME

lofs – loopback virtual file system

CONFIG

options LOFS

SYNOPSIS

#include <sys/mount.h>
mount(MOUNT_LOFS, virtual, flags, dir);

DESCRIPTION

The loopback file system device allows new, virtual file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original file system, however, *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

virtual is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *flags* is either 0 or **M_RDONLY**. The **M_RDONLY** flag forces all accesses in the new name space to be read-only; without it, accesses are the same as for the underlying file system. All other **mount**(2V) flags are preserved from the underlying file systems.

A loopback mount of '/' onto **/tmp/newroot** allows the entire file system hierarchy to appear as if it were duplicated under **/tmp/newroot**, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to '/', or from a pathname relative to **/tmp/newroot** until such time as a file system is mounted in **/tmp/newroot**, or any of its subdirectories.

Loopback mounts of '/' can be performed in conjunction with the **chroot**(2) system call, to provide a complete virtual file system to a process or family of processes.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another file system hierarchy; rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

The standard RC files perform first **4.2** mounts, then **nfs** mounts, during booting. On Sun386i systems, **lo** (loopback) mounts are performed just after **4.2** mounts. **/etc/fstab** files depending on alternate mount orders at boot time will fail to work as expected. Manual modification of **/etc/rc.local** will be needed to make such mount orders work.

WARNINGS

Loopback mounts must be used with care; the potential for confusing users and applications is enormous. A loopback mount entry in **/etc/fstab** must be placed after the mount points of both directories it depends on. This is most easily accomplished by making the loopback mount entry the last in **/etc/fstab**, though see **mount**(8) for further warnings.

SEE ALSO

chroot(2), **mount**(2V), **fstab**(5), **mount**(8)

BUGS

Because only directories can be mounted or mounted on, the structure of a virtual file system can only be modified at directories.

NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

CONFIG — SUN-3, SUN-4 SYSTEMS

MCP

device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88

ALM-2

pseudo-device mcpa64

CONFIG — SUN-3x SYSTEMS

MCP

device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88
device mcp4 at vme32d32 ? csr 0x1040000 flags 0x1ffff priority 4 vector mcpintr 0xa0
device mcp5 at vme32d32 ? csr 0x1050000 flags 0x1ffff priority 4 vector mcpintr 0xa1
device mcp6 at vme32d32 ? csr 0x1060000 flags 0x1ffff priority 4 vector mcpintr 0xa2
device mcp7 at vme32d32 ? csr 0x1070000 flags 0x1ffff priority 4 vector mcpintr 0xa3

ALM-2

pseudo-device mcpa64

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*xy*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION (MCP)

The Sun MCP (Multiprotocol Communications Processor) supports up to four synchronous serial lines in conjunction with SunLink™ Multiple Communication Protocol products.

DESCRIPTION (ALM-2)

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mcp** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mcp0** would treat line **/dev/ttyh2** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* represents the physical board as one of the characters **h**, **i**, **j**, or **k**, and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/ttyh0**, and the sixteenth line on the third board is **/dev/ttyjf**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd*n***, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua*n***, where *n* is the number of the dial-in line.

The **/dev/cua*n*** lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua*n*** line is opened, the corresponding tty line cannot be opened until the **/dev/cua*n*** line is closed; a blocking open will wait until the **/dev/cua*n*** line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd*n*** line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua*n*** line cannot be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttyh0**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

IOCTLS

The standard set of **termio ioctl()** calls are supported by the ALM-2.

If the **CRTSCTS** flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the **CRTSCTS** flag is clear, the state of CTS has no effect. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl()** calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An **open()** on a **/dev/tty*** or a **/dev/cu*** device will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl()** call.
EINTR	The open was interrupted by the delivery of a signal.

DESCRIPTION (PRINTER PORT)

The printer port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

Minor device numbers in the range 64 – 67 access the printer port, and the recommended naming is **/dev/mcpp[0-3]**.

IOCTLS

Various control flags and status bits may be fetched and set on an MCP printer port. The following flags and status bits are supported; they are defined in **sundev/mcpcmd.h**:

MCPRIGNSLCT	0x02	set if interface ignoring SLCT– on open
MCPRDIAG	0x04	set if printer is in self-test mode
MCPRVMEINT	0x08	set if VME bus interrupts enabled
MCPRINTPE	0x10	print message when out of paper
MCPRINTSLCT	0x20	print message when printer offline

MCPRPE	0x40	set if device ready, cleared if device out of paper
MCPRSLCT	0x80	set if device online (Centronics SLCT asserted)

The flags **MCPRINTSLCT, MCPRINTPE,** and **MCPRDIAG** may be changed; the other bits are status bits and may not be changed.

The **ioctl()** calls supported by MCP printer ports are listed below.

MCPIOGPR
: The argument is a pointer to an **unsigned char**. The printer flags and status bits are stored in the **unsigned char** pointed to by the argument.

MCPIOSPR
: The argument is a pointer to an **unsigned char**. The printer flags are set from the **unsigned char** pointed to by the argument.

ERRORS

Normally, the interface only reports the status of the device when attempting an **open**(2V) call. An **open**() on a **/dev/mcpp*** device will fail if:

ENXIO
: The unit being opened does not exist.

EIO
: The device is offline or out of paper.

Bit 17 of the configuration **flags** may be specified to say that the interface should ignore Centronics SLCT– and RDY/PE– when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT– and RDY lines are not asserted during an actual data transfer (as with a **write**(2V) call), no data is transferred.

FILES

/dev/mcpp[0-3]	parallel printer port
/dev/tty[h-k][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S), **ttysoftcar**(8)

DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

mcpa*n*: silo overflow.
: More than *n* characters (*n* very large) have been received by the **mcp** hardware without being read by the software.

*****port *n* supports RS449 interface*****
: Probably an incorrect jumper configuration. Consult the hardware manual.

mcp port *n* receive buffer error
: The **mcp** encountered an error concerning the synchronous receive buffer.

Printer on mcpp*n* is out of paper
Printer on mcpp*n* paper ok
Printer on mcpp*n* is offline
Printer on mcpp*n* online
: Assorted printer diagnostics, if enabled as discussed above.

BUGS

Note: pin 4 is used for hardware flow control on ALM–2 ports 0 through 3. These two pins should *not* be tied together on the ALM end.

NAME

mem, kmem, zero, vme16d16, vme24d16, vme32d16, vme16d32, vme24d32, vme32d32, eeprom, atbus, sbus – main memory and bus I/O space

CONFIG

None; included with standard system.

DESCRIPTION

These devices are special files that map memory and bus I/O space. They may be read, written, seeked and (except for **kmem**) memory-mapped. See **read**(2V), **write**(2V), **mmap**(2), and **directory**(3V).

All Systems

mem is a special file that is an image of the physical memory of the computer. It may be used, for example, to examine (and even to patch) the system.

kmem is a special file that is an image of the kernel virtual memory of the system.

zero is a special file which is a source of private zero pages.

eeprom is a special file that is an image of the EEPROM or NVRAM.

Sun-3 and Sun-4 Systems VMEbus

vme16d16 (also known as **vme16**) is a special file that is an image of VMEbus 16-bit addresses with 16-bit data. **vme16** address space extends from 0 to 64K.

vme24d16 (also known as **vme24**) is a special file that is an image of VMEbus 24-bit addresses with 16-bit data. **vme24** address space extends from 0 to 16 Megabytes. The VME 16-bit address space overlaps the top 64K of the 24-bit address space.

vme32d16 is a special file that is an image of VMEbus 32-bit addresses with 16-bit data.

vme16d32 is a special file that is an image of VMEbus 16-bit addresses with 32-bit data.

vme24d32 is a special file that is an image of VMEbus 24-bit addresses with 32-bit data.

vme32d32 (also known as **vme32**) is a special file that is an image of VMEbus 32-bit addresses with 32-bit data. **vme32** address space extends from 0 to 4 Gigabytes. The VME 24-bit address space overlaps the top 16 Megabytes of the 32-bit address space.

SPARCstation 1 Systems

The **sbus** is represented by a series of entries each of which is an image of a single **sbus** slot. The entries are named **sbus***n*, where *n* is the slot number in hexadecimal. The number of **sbus** slots and the address range within each slot may vary between implementations.

Sun386i Systems

atbus is a special file that is an image of the AT bus space. It extends from 0 to 16 Megabytes.

FILES

/dev/mem
/dev/kmem
/dev/zero
/dev/vme16d16
/dev/vme16
/dev/vme24d16
/dev/vme24
/dev/vme32d16
/dev/vme16d32
/dev/vme24d32
/dev/vme32d32
/dev/vme32
/dev/eeprom
/dev/atbus
/dev/sbus[0-3]

SEE ALSO

mmap(2), **read**(2V), **write**(2V), **directory**(3V)

NAME

mouse – Sun mouse

CONFIG

None; included in standard system.

DESCRIPTION

The **mouse** indirect device provides access to the Sun Workstation mouse. When opened, it redirects operations to the standard mouse device for the workstation (attached either to a CPU serial or parallel port), and pushes the **ms**(4M) and **ttcompat**(4M) STREAMS modules on top of that device.

FILES

/dev/mouse

SEE ALSO

ms(4M), **ttcompat**(4M), **win**(4S), **zs**(4S)

NAME

ms – Sun mouse STREAMS module

CONFIG

pseudo-device*ms***n**

SYNOPSIS

#include <sys/types.h>
#include <sys/time.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sundev/vuid_event.h>
#include <sundev/msio.h>
ioctl(fd, I_PUSH, "ms");

DESCRIPTION

The **ms** STREAMS module processes byte streams generated by mice attached to a CPU serial or parallel port. When this module is pushed onto a stream, it sends a **TCSETSF** ioctl downstream, setting the baud rate to 1200 baud and the character size to 8 bits, and enabling the receiver. All other flag words are cleared. It assumes only that the **termios**(3V) functions provided by the **zs**(4S) driver are supported; no other functions need be supported.

The mouse is expected to generate a stream of bytes encoding mouse motions and changes in the state of the buttons.

Each mouse sample in the byte stream consists of three bytes: the first byte gives the button state with value 0x87|~*but*, where *but* is the low three bits giving the mouse buttons, where a 0 (zero) bit means that a button is pressed, and a 1 (one) bit means a button is not pressed. Thus if the left button is down the value of this sample is 0x83, while if the right button is down the byte is 0x86.

The next two bytes of each sample give the *x* and *y* deltas of this sample as signed bytes. The mouse uses a lower-left coordinate system, so moves to the right on the screen yield positive *x* values and moves down the screen yield negative *y* values.

The beginning of a sample is identifiable because the delta's are constrained to not have values in the range 0x80-0x87.

A stream with **ms** pushed onto it can be used as a device that emits *firm_events* as specified by the protocol of a *Virtual User Input Device*. It understands **VUIDSFORMAT, VUIDGFORMAT, VUIDSADDR** and **VUIDGADDR** ioctls (see reference below).

IOCTLS

ms responds to the following *ioctl*s, as defined in **<sundev/msio.h>** and **<sundev/vuid_event.h>**. All other *ioctl*s are passed downstream. As **ms** sets the parameters of the serial port when it is opened, no **termios**(3V) *ioctl*s should be performed on a stream with **ms** on it, as **ms** expects the device parameters to remain as it set them.

The **MSIOGETPARMS** and **MSIOSETPARMS** calls use a structure of type **Ms_parms**, which is a structure defined in **<sundev/msio.h>**:

```
typedef struct {
int        jitter_thresh;
int        speed_law;
int        speed_limit;
}          Ms_parms;
```

jitter_thresh is the "jitter threshold" of the mouse. Motions of fewer than *jitter_thresh* units along both axes that occur in less than 1/12 second are treated as "jitter" and ignored. Thus, if the mouse moves fewer than *jitter_thresh units* and then moves back to its original position in less than 1/12 of a second, the motion is considered to be "noise" and ignored. If it moves fewer than *jitter_thresh* units and continues to move so that it has not returned to its original position after 1/12 of a second, the motion is considered to be real and is reported.

speed_law indicates whether extremely large motions are to be ignored. If it is 1, a "speed limit" is applied to mouse motions; motions along either axis of more than *speed_limit* units are discarded.

Note: these parameters are global; if they are set for any mouse on a workstation, they apply to any other mice attached to that workstation as well.

VUIDSFORMAT
VUIDGFORMAT
VUIDSADDR
VUIDGADDR These are standard *Virtual User Input Device ioctl*s. See *SunView System Programmer's Guide* for a description of their operation.

MSIOGETPARMS The argument is a pointer to a **Ms_parms**. The current mouse parameters are stored in that structure.

MSIOSETPARMS The argument is a pointer to a **ms_parms**. The current mouse parameters are set from the values in that structure.

SEE ALSO

mouse(4S), **termios**(3V), **win**(4S), **zs**(4S)

SunView System Programmer's Guide

NAME

mti – Systech MTI-800/1600 multi-terminal interface

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

device mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88
device mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89
device mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a
device mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*xy*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION

The Systech MTI card provides 8 (MTI-800) or 16 (MTI-1600) serial communication lines with modem control. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mti** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed on top of the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mti0** would treat line **/dev/tty02** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* is the physical board number (0 – 3), and *Y* is the line number on the board as a single hexadecimal digit. Thus the first line on the first board is **/dev/tty00**, and the sixteenth line on the third board is **/dev/tty2f**.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line can not be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua***n* line can not be opened. This allows a modem to be attached to for example, **/dev/ttyd0** (renamed from **/dev/tty00**) and used for dial-in (by enabling the line for login in **/etc/ttytab**) and also used for dial-out (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

WIRING

The Systech requires the CTS modem control signal to operate. If the device does not supply CTS then RTS should be jumpered to CTS at the distribution panel (short pins 4 to 5). Also, the CD (carrier detect) line does not work properly. When connecting a modem, the modem's CD line should be wired to DSR, which the software will treat as carrier detect.

IOCTLS

The standard set of **termio ioctl( )** calls are supported by **mti**.

The state of the **CRTSCTS** flag in the **c_cflag** word has no effect; no output will be generated unless CTS is high. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl( )** calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided; however, as described above, the DSR line is treated as CD and the CD line is ignored.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed. The baud rates **B200** and **B38400** are not supported by the hardware; **B200** selects 2000 baud, and **B38400** selects 7200 baud.

ERRORS

An **open( )** will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl( )** call.
EINTR	The open was interrupted by the delivery of a signal.

FILES

/dev/tty[0-3][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dial-in tty lines
/dev/cua[0-9a-f]	dial-out tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mcp**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S), **ttysoftcar**(8)

DIAGNOSTICS

Most of these diagnostics "should never happen" and their occurrence usually indicates problems elsewhere in the system.

mti*n*, *n*: silo overflow.
More than 512 characters have been received by the **mti** hardware without being read by the software. Extremely unlikely to occur.

mti*n*: read error code <*n*>. Probable hardware fault
The **mti** returned the indicated error code. See the MTI manual.

mti*n*: DMA output error.
The **mti** encountered an error while trying to do DMA output.

mti*n*: impossible response *n*.
The **mti** returned an error it could not understand.

NAME

mtio – general magnetic tape interface

SYNOPSIS

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mtio.h>

DESCRIPTION

1/2", 1/4" and 8 mm magnetic tape drives all share the same general character device interface.

There are two types of tape records: data records and end-of-file (EOF) records. EOF records are also known as tape marks and file marks. A record is separated by interrecord (or tape) gaps on a tape.

End-of-recorded-media (EOM) is indicated by two EOF marks on 1/2" tape; by one on 1/4" and 8 mm cartridge tapes.

1/2" Reel Tape

Data bytes are recorded in parallel onto the 9–track tape. The number of bytes in a physical record varies between 1 and 65535 bytes.

The recording formats available (check specific tape drive) are 800 BPI, 1600 BPI, and 6250 BPI, and data compression. Actual storage capacity is a function of the recording format and the length of the tape reel. For example, using a 2400 foot tape, 20 MB can be stored using 800 BPI, 40 MB using 1600 BPI, 140 MB using 6250 BPI, or up to 700 MB using data compression.

1/4" Cartridge Tape

Data is recorded serially onto 1/4" cartridge tape. The number of bytes per record is determined by the physical record size of the device. The I/O request size must be a multiple of the physical record size of the device. For QIC–11, QIC–24, and QIC–150 tape drives the block size is 512 bytes.

The records are recorded on tracks in a serpentine motion. As one track is completed, the drive switches to the next and begins writing in the opposite direction, eliminating the wasted motion of rewinding. Each file, including the last, ends with one file mark.

Storage capacity is based on the number of tracks the drive is capable of recording. For example, 4–track drives can only record 20 MB of data on a 450 foot tape; 9–track drives can record up to 45 MB of data on a tape of the same length. QIC–11 is the only tape format available for 4–track tape drives. In contrast, 9–track tape drives can use either QIC–24 or QIC–11. Storage capacity is not appreciably affected by using either format. QIC–24 is preferable to QIC–11 because it records a reference signal to mark the position of the first track on the tape, and each block has a unique block number.

The QIC–150 tape drives require DC–6150 (or equivalent) tape cartridges for writing. However, they can read other tape cartridges in QIC–11, QIC–24, QIC–120, or QIC–150 tape formats.

8 mm Cartridge Tape

Data is recorded serially onto 8 mm helical scan cartridge tape. The number of bytes in a physical record varies between 1 and 65535 bytes. Currently one density is available.

Read Operation

read(2V) reads the next record on the tape. The record size is passed back as the number of bytes read, provided it is no greater than the number requested. When a tape mark is read, a zero byte count is returned; another read will fetch the first record of the next tape file. Two successive reads returning zero byte counts indicate the EOM. No further reading should be performed past the EOM.

Fixed-length I/O tape devices require the number of bytes read to be a multiple of the physical record size. For example, 1/4" cartridge tape devices only read multiples of 512 bytes. If the blocking factor is greater than 64512 bytes (minphys limit), fixed-length I/O tape devices read multiple records.

Tape devices which support variable-length I/O operations, such as 1/2" and 8mm tape, may read a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver reads multiple records to satisfy the request. These multiple records are limited to 65534 bytes.

Write Operation

write(2V) writes the next record on the tape. The record has the same length as the given buffer.

Writing is allowed on 1/4" tape at either the beginning of tape or after the last written file on the tape.

Writing is not so restricted on 1/2" and 8 mm cartridge tape. Care should be used when appending files onto 1/2" reel tape devices, since an extra file mark is appended after the last file to mark the EOM. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the EOM ioctl is provided. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

Fixed-length I/O tape devices require the number of bytes written to be a multiple of the physical record size. For example, 1/4" cartridge tape devices only write multiples of 512 bytes. Fixed-length I/O tape devices write multiple records if the blocking factor is greater than 64512 bytes (minphys limit). These multiple writes are limited to 64512 bytes. For example, if a write request is issued for 65536 bytes using a 1/4" cartridge tape, two writes are issued; the first for 64512 bytes and the second for 1024 bytes.

Tape devices which support variable-length I/O operations, such as 1/2" and 8mm tape, may write a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver writes multiple records to satisfy the request. These multiple records are limited to 65534 bytes. As an example, if a write request for 65540 bytes is issued using 1/2" reel tape, two records are written; one for 65534 bytes followed by one for 6 bytes.

EOT handling on write is different among the various devices; see the appropriate device manual page. Reading past EOT is transparent to the user.

Seeks are ignored in tape I/O.

Close Operation

Magnetic tapes are rewound when closed, except when the "no-rewind" devices have been specified. The names of no-rewind device files use the letter **n** as the beginning of the final component. The no-rewind version of **/dev/rmt0** is **/dev/nrmt0**.

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device was specified, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning ioctl, like rewind, is issued after writing, a file mark is written before repositioning the tape.

Note: for 1/2" reel tape devices, two file marks are written to mark the EOM before rewinding or performing a file positioning ioctl. If the user wrote a file mark before closing a 1/2" reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly. If the non-rewinding **xt** device was specified, two file marks are written and the tape is left positioned between the two so that the second one is overwritten on a subsequent **open**(2V) and **write**(2V). For performance reasons, the **st** driver postpones writing the second tape mark until just before a file positioning ioctl is issued (for example, rewind). This means that the user must not manually rewind the tape because the tape will be missing the second tape mark which marks EOM.

If no data was written and the driver was opened for WRITE-ONLY access, a file mark is written thus creating a null file.

Ioctls

Not all devices support all ioctls. The driver returns an ENOTTY error on unsupported ioctls.

The following structure definitions for magnetic tape ioctl commands are from **<sys/mtio.h>**:

```
/* structure for MTIOCTOP – magnetic tape operation command */
struct  mtop  {
        short   mt_op;          /* operation */
        daddr_t mt_count;       /* number of operations */
};
```

The following ioctls are supported:

MTWEOF	write an end-of-file record
MTFSF	forward space over file mark
MTBSF	backward space over file mark (1/2", 8 mm only)
MTFSR	forward space to inter-record gap
MTBSR	backward space to inter-record gap
MTREW	rewind
MTOFFL	rewind and take the drive offline
MTNOP	no operation, sets status only
MTRETEN	retension the tape (cartridge tape only)
MTERASE	erase the entire tape and rewind
MTEOM	position to EOM
MTNBSF	backward space file to beginning of file

```
/* structure for MTIOCGET – magnetic tape get status command */
struct  mtget {
        short   mt_type;                /* type of magtape device */
/* the following two registers are device dependent */
        short   mt_dsreg;               /* "drive status" register */
        short   mt_erreg;               /* "error" register */
/* optional error info. */
        daddr_t mt_resid;               /* residual count */
        daddr_t mt_fileno;              /* file number of current position */
        daddr_t mt_blkno;               /* block number of current position */
        u_short mt_flags;
        short   mt_bf;                  /* optimum blocking factor */
};
```

When spacing forward over a record (either data or EOF), the tape head is positioned in the tape gap between the record just skipped and the next record. When spacing forward over file marks (EOF records), the tape head is positioned in the tape gap between the next EOF record and the record that follows it.

When spacing backward over a record (either data or EOF), the tape head is positioned in the tape gap immediately preceding the tape record where the tape head is currently positioned. When spacing backward over file marks (EOF records), the tape head is positioned in the tape gap preceding the EOF. Thus the next read would fetch the EOF.

Note, the following features are unique to the **st** driver: record skipping does not go past a file mark; file skipping does not go past the EOM. Both the **st** and **xt** drivers stop upon encountering EOF during a record skipping command, but leave the tape positioned differently. For example, after an MTFSR <huge number> command the **st** driver leaves the tape positioned *before* the EOF. After the same command, the **xt** driver leaves the tapes positioned *after* the EOF. Consequently on the next read, the **xt** driver fetches the first record of the next file whereas the **st** driver fetches the EOF. A related **st** feature is that EOFs remain pending until the tape is closed. For example, a program which first reads all the records of a file up to and including the EOF and then performs an MTFSF command will leave the tape positioned just after that same EOF, rather than skipping the next file.

The MTNBSF and MTFSF operations are inverses. Thus, an MTFSF "–1" is equivalent to an MTNBSF "1". An MTNBSF "0" is the same as MTFSF "0"; both position the tape device to the beginning of the current file.

MTBSF moves the tape backwards by file marks. The tape position will end on the beginning of tape side of the desired file mark.

MTBSR and MTFSR operations perform much like space file operations, except that they move by records instead of files. Variable-length I/O devices (1/2" reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4" cartridge tape, for example, spaces 512 byte physical records. The status ioctl residual count contains the number of files or records not skipped.

MTOFFL rewinds and, if appropriate, takes the device offline by unloading the tape. The tape must be inserted before the tape device can be used again.

MTRETEN The retension ioctl only applies to 1/4" cartridge tape devices. It is used to restore tape tension improving the tape's soft error rate after extensive start-stop operations or long-term storage.

MTERASE rewinds the tape, erases it completely, and returns to the beginning of tape.

MTEOM positions the tape at a location just after the last file written on the tape. For 1/4" cartridge and 8 mm tape, this is after the last file mark on the tape. For 1/2" reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.

Note the difference between MTBSF (backspace over file mark) and MTNBSF (backspace file to beginning of file). The former moves the tape backward until it crosses an EOF mark, leaving the tape positioned *before* the file mark. The latter leaves the tape positioned *after* the file mark. Hence, "MTNBSF n" is equivalent to "MTBSF (n+1)" followed by "MTFSF 1". 1/4 " cartridge tape devices do not support MTBSF.

The MTIOCGET get status ioctl call returns the drive id (*mt_type*), sense key error (*mt_erreg*), file number (*mt_fileno*), optimum blocking factor (*mt_bf*) and record number (*mt_blkno*) of the last error. The residual count (*mt_resid*) is set to the number of bytes not transferred or files/records not spaced. The flags word (*mt_flags*) contains information such as whether the device is SCSI, whether it is a reel device and whether the device supports absolute file positioning.

EXAMPLES

Suppose you have written 3 files to the non-rewinding 1/2" tape device, **/dev/nrmt0**, and that you want to go back and **dd**(1) the second file off the tape. The commands to do this are:

```
mt –f /dev/nrmt0 bsf 3
mt –f /dev/nrmt0 fsf 1
dd if=/dev/nrmt0
```

To accomplish the same tape positioning in a C program, followed by a get status ioctl:

```
struct mtop mt_command;
struct mtget mt_status;

mt_command.mt_op = MTBSF;
mt_command.mt_count = 3;
ioctl(fd, MTIOCTOP, &mt_command);
mt_command.mt_op = MTFSF;
mt_command.mt_count = 1;
ioctl(fd, MTIOCTOP, &mt_command);
ioctl(fd, MTIOCGET, (char *)&mt_status);
```

or

```
struct mtop mt_command;
struct mtget mt_status;

mt_command.mt_op = MTNBSF;
mt_command.mt_count = 2;
ioctl(fd, MTIOCTOP, &mt_command);
ioctl(fd, MTIOCGET, (char *)&mt_status);
```

FILES

/dev/rmt*
/dev/rst*
/dev/rar*
/dev/nrmt*
/dev/nrst*
/dev/nrar*

SEE ALSO

dd(1), **mt**(1), **tar**(1), **read**(2V), **write**(2V), **ar**(4S), **st**(4S), **tm**(4S), **xt**(4S)

1/4 Inch Tape Drive Tutorial

WARNINGS

Avoid the use of device files **/dev/rmt4** and **/dev/rmt12**, as they are going away in a future release.

NAME

nfs, NFS – network file system

CONFIG

options NFS

DESCRIPTION

The Network File System, or NFS, allows a client workstation to perform transparent file access over the network. Using it, a client workstation can operate on files that reside on a variety of servers, server architectures and across a variety of operating systems. Client file access calls are converted to NFS protocol requests, and are sent to the server system over the network. The server receives the request, performs the actual file system operation, and sends a response back to the client.

The Network File System operates in a stateless fashion using remote procedure (RPC) calls built on top of external data representation (XDR) protocol. These protocols are documented in *Network Programming* The RPC protocol provides for version and authentication parameters to be exchanged for security over the network.

A server can grant access to a specific filesystem to certain clients by adding an entry for that filesystem to the server's **/etc/exports** file and running **exportfs**(8).

A client gains access to that filesystem with the **mount**(2V) system call, which requests a file handle for the filesystem itself. Once the filesystem is mounted by the client, the server issues a file handle to the client for each file (or directory) the client accesses or creates. If the file is somehow removed on the server side, the file handle becomes stale (dissociated with a known file).

A server may also be a client with respect to filesystems it has mounted over the network, but its clients cannot gain access to those filesystems. Instead, the client must mount a filesystem directly from the server on which it resides.

The user ID and group ID mappings must be the same between client and server. However, the server maps uid 0 (the super-user) to uid –2 before performing access checks for a client. This inhibits super-user privileges on remote filesystems. This may be changed by use of the "anon" export option. See **exportfs**(8).

NFS-related routines and structure definitions are described in *Network Programming.*

ERRORS

Generally physical disk I/O errors detected at the server are returned to the client for action. If the server is down or inaccessible, the client will see the console message:

NFS server *host* **not responding still trying.**

Depending on whether the file system has been mounted "hard" or "soft" (see **mount**(8)), the client will either continue (forever) to resend the request until it receives an acknowledgement from the server, or return an error to user-level. For hard mounts, this means the server can crash or power down and come back up without any special action required by the client. If the "intr" mount option was not specified, a client process requesting I/O will block and remain insensitive to signals, sleeping inside the kernel at **PRIBIO** until the request is satisfied.

FILES

/etc/exports

SEE ALSO

mount(2V), **exports**(5), **fstab**(5), **fstab**(5), **exportfs**(8), **mount**(8), **nfsd**(8), **sticky**(8)

Network Programming

BUGS

When a file that is opened by a client is unlinked (by the server), a file with a name of the form **.nfs***XXX* (where *XXX* is a number) is created by the client. When the open file is closed, the **.nfs***XXX* file is removed. If the client crashes before the file can be closed, the **.nfs***XXX* file is not removed.

NFS servers usually mark their clients' swap files specially to avoid being required to sync their inodes to disk before returning from writes. See **sticky**(8).

NAME

nit – Network Interface Tap

CONFIG

pseudo-device clone
pseudo-device snit
pseudo-device pf
pseudo-device nbuf

SYNOPSIS

```
#include <sys/file.h>
#include <sys/ioctl.h>
#include <net/nit_pf.h>
#include <net/nit_buf.h>

        fd = open("/dev/nit", mode);
        ioctl(fd, I_PUSH, "pf");
        ioctl(fd, I_PUSH, "nbuf");
```

DESCRIPTION

NIT (the Network Interface Tap) is a facility composed of several STREAMS modules and drivers. These components collectively provide facilities for constructing applications that require link-level network access. Examples of such applications include **rarpd**(8C), which is a user-level implementation of the Reverse ARP protocol, and **etherfind**(8C), which is a network monitoring and trouble-shooting program.

NIT consists of several components that are summarized below. See their Reference Manual entries for detailed information about their specification and operation.

nit_if(4M) This component is a STREAMS device driver that interacts directly with the system's Ethernet drivers. After opening an instance of this device it must be bound to a specific Ethernet interface before becoming usable. Subsequently, **nit_if** transcribes packets arriving on the interface to the read side of its associated stream and delivers messages reaching it on the write side of its stream to the raw packet output code for transmission over the interface.

nit_pf(4M) This module provides packet-filtering services, allowing uninteresting incoming packets to be discarded with minimal loss of efficiency. It passes through unaltered all outgoing messages (those on the stream's write side).

nit_buf(4M) This module buffers incoming messages into larger aggregates, thereby reducing the overhead incurred by repeated **read**(2V) system calls.

NIT clients mix and match these components, based on their particular requirements. For example, the reverse ARP daemon concerns itself only with packets of a specific type and deals with low traffic volumes. Thus, it uses **nit_if** for access to the network and **nit_pf** to filter out all incoming packets except reverse ARP packets, but omits the **nit_buf** buffering module since traffic is not high enough to justify the additional complexity of unpacking buffered packets. On the other hand, the **etherd**(8C) program, which collects Ethernet statistics for **traffic**(1C) to display, must examine every packet on the network. Therefore, it omits the **nit_pf** module, since there is nothing it wishes to screen out, and includes the **nit_buf** module, since most networks have very heavy aggregate packet traffic.

EXAMPLES

The following code fragments outline how to program against parts of the NIT interface. For the sake of brevity, all error-handling code has been elided.

initdevice comes from **etherfind** and sets up its input stream configuration.

```
initdevice(if_flags, snaplen, chunksize)
        u_long  if_flags,
                snaplen,
                chunksize;
```

```
{
        struct strioctl     si;
        struct ifreq        ifr;
        struct timeval      timeout;

        if_fd = open(NIT_DEV, O_RDONLY);

        /* Arrange to get discrete messages from the stream. */
        ioctl(if_fd, I_SRDOPT, (char *)RMSGD);

        si.ic_timout = INFTIM;

        /* Push and configure the buffering module. */
        ioctl(if_fd, I_PUSH, "nbuf");

        timeout.tv_sec = 1;
        timeout.tv_usec = 0;
        si.ic_cmd = NIOCSTIME;
        si.ic_len = sizeof timeout;
        si.ic_dp = (char *)&timeout;
        ioctl(if_fd, I_STR, (char *)&si);

        si.ic_cmd = NIOCSCHUNK;
        si.ic_len = sizeof chunksize;
        si.ic_dp = (char *)&chunksize;
        ioctl(if_fd, I_STR, (char *)&si);

        /* Configure the nit device, binding it to the proper
          underlying interface, setting the snapshot length,
          and setting nit_if-level flags. */
        strncpy(ifr.ifr_name, device, sizeof ifr.ifr_name);
        ifr.ifr_name[sizeof ifr.ifr_name - 1] = '\0';
        si.ic_cmd = NIOCBIND;
        si.ic_len = sizeof ifr;
        si.ic_dp = (char *)&ifr;
        ioctl(if_fd, I_STR, (char *)&si);

        if (snaplen > 0) {
                si.ic_cmd = NIOCSSNAP;
                si.ic_len = sizeof snaplen;
                si.ic_dp = (char *)&snaplen;
                ioctl(if_fd, I_STR, (char *)&si);
        }

        if (if_flags != 0) {
                si.ic_cmd = NIOCSFLAGS;
                si.ic_len = sizeof if_flags;
                si.ic_dp = (char *)&if_flags;
                ioctl(if_fd, I_STR, (char *)&si);
        }

        /* Flush the read queue, to get rid of anything that accumulated
          before the device reached its final configuration. */
        ioctl(if_fd, I_FLUSH, (char *)FLUSHR);
}
```

Here is the skeleton of the packet reading loop from **etherfind**. It illustrates how to cope with dismantling the headers the various NIT components glue on.

```
while ((cc = read(if_fd, buf, chunksize)) >= 0) {
        register u_char     *bp = buf,
                            *bufstop = buf + cc;

        /* Loop through each message in the chunk. */
        while (bp < bufstop) {
                register u_char             *cp = bp;
                struct nit_bufhdr  *hdrp;
                struct timeval              *tvp = NULL;
                u_long                      drops = 0;
                u_long                      pktlen;

                /* Extract information from the successive objects
                  embedded in the current message. Which ones we
                  have depends on how we set up the stream (and
                  therefore on what command line flags were set).

                  If snaplen is positive then the packet was truncated
                  before the buffering module saw it, so we must
                  obtain its length from the nit_if-level nit_iflen
                  header. Otherwise the value in *hdrp suffices. */
                hdrp = (struct nit_bufhdr *)cp;
                cp += sizeof *hdrp;
                if (tflag) {
                        struct nit_iftime    *ntp;

                        ntp = (struct nit_iftime *)cp;
                        cp += sizeof *ntp;

                        tvp = &ntp->nh_timestamp;
                }
                if (dflag) {
                        struct nit_ifdrops  *ndp;

                        ndp = (struct nit_ifdrops *)cp;
                        cp += sizeof *ndp;

                        drops = ndp->nh_drops;
                }
                if (snaplen > 0) {
                        struct nit_iflen      *nlp;

                        nlp = (struct nit_iflen *)cp;
                        cp += sizeof *nlp;

                        pktlen = nlp->nh_pktlen;
                }
                else
                        pktlen = hdrp->nhb_msglen;

                sp = (struct sample *)cp;
                bp += hdrp->nhb_totlen;

                /* Process the packet. */
        }
}
```

FILES

/dev/nit clone device instance referring to **nit_if**

SEE ALSO

traffic(1C), **read**(2V), **nit_if**(4M), **nit_pf**(4M), **nit_buf**(4M), **etherd**(8C), **etherfind**(8C), **rarpd**(8C)

NAME

nit_buf – STREAMS NIT buffering module

CONFIG

pseudo-device nbuf

SYNOPSIS

#include <sys/ioctl.h>
#include <net/nit_buf.h>
ioctl(fd, I_PUSH, "nbuf");

DESCRIPTION

nit_buf is a STREAMS module that buffers incoming messages, thereby reducing the number of system calls and associated overhead required to read and process them. Although designed to be used in conjunction with the other components of NIT (see **nit**(4P)), **nit_buf** is a general-purpose module and can be used anywhere STREAMS input buffering is required.

Read-side Behavior

nit_buf collects incoming **M_DATA** and **M_PROTO** messages into *chunks*, passing each chunk upward when either the chunk becomes full or the current read timeout expires. When a message arrives, it is processed in two steps. First, the message is prepared for inclusion in a chunk, and then it is added to the current chunk. The following paragraphs discuss each step in turn.

Upon receiving a message from below, **nit_buf** immediately converts all leading **M_PROTO** blocks in the message to **M_DATA** blocks, altering only the message type field and leaving the contents alone. It then prepends a header to the converted message. This header is defined as follows.

```
struct nit_bufhdr {
        u_int   nhb_msglen;
        u_int   nhb_totlen;
};
```

The first field of this header gives the length in bytes of the converted message. The second field gives the distance in bytes from the start of the message in the current chunk (described below) to the start of the next message in the chunk; the value reflects any padding necessary to insure correct data alignment for the host machine and includes the length of the header itself.

After preparing a message, **nit_buf** attempts to add it to the end of the current chunk, using the chunk size and timeout values to govern the addition. (The chunk size and timeout values are set and inspected using the **ioctl** calls described below.) If adding the new message would make the current chunk grow larger than the chunk size, **nit_buf** closes off the current chunk, passing it up to the next module in line, and starts a new chunk, seeding it with a zero-length message. If adding the message would still make the current chunk overflow, the module passes it upward in an over-size chunk of its own. Otherwise, the module concatenates the message to the end of the current chunk.

To ensure that messages do not languish forever in an accumulating chunk, **nit_buf** maintains a read timeout. Whenever this timeout expires, the module closes off the current chunk, regardless of its length, and passes it upward; if no incoming messages have arrived, the chunk passed upward will have zero length. Whenever the module passes a chunk upward, it restarts the timeout period. These two rules insure that **nit_buf** minimizes the number of chunks it produces during periods of intense message activity and that it periodically disposes of all messages during slack intervals.

nit_buf handles other message types as follows. Upon receiving an **M_FLUSH** message specifying that the read queue be flushed, the module does so, clearing the currently accumulating chunk as well, and passes the message on to the module or driver above. It passes all other messages through unaltered to its upper neighbor.

Write-side Behavior

nit_buf intercepts **M_IOCTL** messages for the *ioctl*s described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, the module does so and passes the message on to the module or driver below. The module passes all other messages through unaltered to its lower neighbor.

IOCTLS

nit_buf responds to the following *ioctl*s.

NIOCSTIME Set the read timeout value to the value referred to by the *struct timeval* pointer given as argument. Setting the timeout value to zero has the side-effect of forcing the chunk size to zero as well, so that the module will pass all incoming messages upward immediately upon arrival.

NIOCGTIME Return the read timeout in the *struct timeval* pointed to by the argument. If the timeout has been cleared with the **NIOCCTIME** *ioctl*, return with an ERANGE error.

NIOCCTIME Clear the read timeout, effectively setting its value to infinity.

NIOCSCHUNK Set the chunk size to the value referred to by the *u_int* pointer given as argument.

NIOCGCHUNK Return the chunk size in the *u_int* pointed to by the argument.

WARNING

The module name ''nbuf'' used in the system configuration file and as argument to the **I_PUSH ioctl** is provisional and subject to change.

SEE ALSO

nit(4P), **nit_if**(4M), **nit_pf**(4M)

NAME

nit_if – STREAMS NIT device interface module

CONFIG

pseudo-device snit

SYNOPSIS

#include <sys/file.h>
open("/dev/nit", mode);

DESCRIPTION

nit_if is a STREAMS pseudo-device driver that provides STREAMS access to network interfaces. It is designed to be used in conjunction with the other components of NIT (see **nit**(4P)), but can be used by itself as a raw STREAMS network interface.

nit_if is an exclusive-open device that is intended to be opened indirectly through the clone device; **/dev/nit** is a suitable instance of the clone device. Before the stream resulting from opening an instance of **nit_if** may be used to read or write packets, it must first be bound to a specific network interface, using the **NIOCSBIND** ioctl described below.

Read-side Behavior

nit_if copies leading prefixes of selected packets from its associated network interface and passes them up the stream. If the **NI_PROMISC** flag is set, it passes along all packets; otherwise it passes along only packets addressed to the underlying interface.

The amount of data copied from a given packet depends on the current *snapshot length*, which is set with the **NIOCSSNAP** ioctl described below.

Before passing each packet prefix upward, **nit_if** optionally prepends one or more headers, as controlled by the state of the flag bits set with the **NIOCSFLAGS** *ioctl*. The driver collects headers into **M_PROTO** message blocks, with the headers guaranteed to be completely contained in a single message block, whereas the packet itself goes into one or more **M_DATA** message blocks.

Write-side Behavior

nit_if accepts packets from the module above it in the stream and relays them to the associated network interface for transmission. Packets must be formatted with the destination address in a leading **M_PROTO** message block, followed by the packet itself, complete with link-level header, in a sequence of **M_DATA** message blocks. The destination address must be expressed as a '**struct sockaddr**' whose *sa_family* field is **AF_UNSPEC** and whose *sa_data* field is a copy of the link-level header. (See **sys/socket.h** for the definition of this structure.) If the packet does not conform to this format, an **M_ERROR** message with EINVAL will be sent upstream.

nit_if processes **M_IOCTL** messages as described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, **nit_if** does so and transfers the message to the read side of the stream. It discards all other messages.

IOCTLS

nit_if responds to the following *ioctls*, as defined in **net/nit_if.h.** It generates an **M_IOCNAK** message for all others, returning this message to the invoker along the read side of the stream.

SIOCGIFADDR

SIOCADDMULTI

SIOCDELMULTI — **nit_if** passes these ioctls on to the underlying interface's driver and returns its response in a '**struct ifreq**' instance, as defined in **net/if.h**. (See the description of this ioctl in **if**(4N) for more details.)

NIOCBIND — This ioctl attaches the stream represented by its first argument to the network interface designated by its third argument, which should be a pointer to an *ifreq* structure whose *ifr_name* field names the desired interface. See **net/if.h** for the definition of this structure.

NIOCSSNAP Set the current snapshot length to the value given in the *u_long* pointed to by the *ioctl*'s final argument. **nit_if** interprets a snapshot length value of zero as meaning infinity, so that it will copy all selected packets in their entirety. It constrains positive snapshot lengths to be at least the length of an Ethernet header, so that it will pass at least the link-level header of all selected packets to its upstream neighbor.

NIOCGSNAP Returns the current snapshot length for this device instance in the *u_long* pointed to by the *ioctl*'s final argument.

NIOCSFLAGS **nit_if** recognizes the following flag bits, which must be given in the *u_long* pointed to by the *ioctl*'s final argument. This set may be augmented in future releases. All but the **NI_PROMISC** bit control the addition of headers that precede the packet body. These headers appear in the order given below, with the last-mentioned enabled header adjacent to the packet body.

- **NI_PROMISC** Requests that the underlying interface be set into promiscuous mode and that all packets that the interface receives be passed up through the stream. **nit_if** only honors this bit for the super-user.
- **NI_TIMESTAMP** Prepend to each selected packet a header containing the packet arrival time expressed as a '**struct timeval**'.
- **NI_DROPS** Prepend to each selected packet a header containing the cumulative number of packets that this instance of **nit_if** has dropped because of flow control requirements or resource exhaustion. The header value is expressed as a *u_long*. Note: it accounts only for events occurring within **nit_if**, and does not count packets dropped at the network interface level or by upstream modules.
- **NI_LEN** Prepend to each selected packet a header containing the packet's original length (including link-level header), as it was before being trimmed to the snapshot length. The header value is expressed as a *u_long*.

NIOCGFLAGS Returns the current state of the flag bits for this device instance in the *u_long* pointed to by the *ioctl*'s final argument.

FILES

/dev/nit clone device instance referring to **nit_if** device

net/nit_if.h header file containing definitions for the *ioctl*s and packet headers described above.

SEE ALSO

clone(4), **nit**(4P), **nit_buf**(4M), **nit_pf**(4M)

NAME

nit_pf – STREAMS NIT packet filtering module

CONFIG

pseudo-device pf

SYNOPSIS

#include <sys/ioctl.h>
#include <net/nit_pf.h>

ioctl(*fd*, I_PUSH, "pf");

DESCRIPTION

nit_pf is a STREAMS module that subjects messages arriving on its read queue to a packet filter and passes only those messages that the filter accepts on to its upstream neighbor. Such filtering can be very useful for user-level protocol implementations and for networking monitoring programs that wish to view only specific types of events.

Read-side Behavior

nit_pf applies the current packet filter to all **M_DATA** and **M_PROTO** messages arriving on its read queue. The module prepares these messages for examination by first skipping over all leading **M_PROTO** message blocks to arrive at the beginning of the message's data portion. If there is no data portion, **nit_pf** accepts the message and passes it along to its upstream neighbor. Otherwise, the module ensures that the part of the message's data that the packet filter might examine lies in contiguous memory, calling the *pullupmsg* utility routine if necessary to force contiguity. (Note: this action destroys any sharing relationships that the subject message might have had with other messages.) Finally, it applies the packet filter to the message's data, passing the entire message upstream to the next module if the filter accepts, and discarding the message otherwise. See PACKET FILTERS below for details on how the filter works.

If there is no packet filter yet in effect, the module acts as if the filter exists but does nothing, implying that all incoming messages are accepted. IOCTLS below describes how to associate a packet filter with an instance of **nit_pf**.

nit_pf handles other message types as follows. Upon receiving an M_FLUSH message specifying that the read queue be flushed, the module does so, and passes the message on to its upstream neighbor. It passes all other messages through unaltered to its upper neighbor.

Write-side Behavior

nit_pf intercepts **M_IOCTL** messages for the *ioctl* described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, the module does so and passes the message on to the module or driver below. The module passes all other messages through unaltered to its lower neighbor.

IOCTLS

nit_pf responds to the following *ioctl*.

NIOCSETF This *ioctl* directs the module to replace its current packet filter, if any, with the filter specified by the '**struct packetfilt**' pointer named by its final argument. This structure is defined in **<net/packetfilt.h>** as

```
struct packetfilt {
      u_char   Pf_Priority;    /* priority of filter */
      u_char   Pf_FilterLen;   /* # of cmds in list */
      u_short  Pf_Filter[ENMAXFILTERS];
                               /* filter command list */
};
```

The *Pf_Priority* field is included only for compatibility with other packet filter implementations and is otherwise ignored. The packet filter itself is specified in the *Pf_Filter* array as a sequence of two-byte commands, with the *Pf_FilterLen* field giving the number of commands in the sequence. This implementation restricts the maximum number of commands in a filter (**ENMAXFILTERS**) to 40. The next section describes the available commands and their semantics.

PACKET FILTERS

A packet filter consists of the filter command list length (in units of *u_shorts*), and the filter command list itself. (The priority field mentioned above is ignored in this implementation.) Each filter command list specifies a sequence of actions that operate on an internal stack of *u_shorts* ("shortwords"). Each shortword of the command list specifies one of the actions ENF_PUSHLIT, ENF_PUSHZERO, or ENF_PUSHWORD+*n*, which respectively push the next shortword of the command list, zero, or shortword *n* of the subject message on the stack, and a binary operator from the set { ENF_EQ, ENF_NEQ, ENF_LT, ENF_LE, ENF_GT, ENF_GE, ENF_AND, ENF_OR, ENF_XOR } which then operates on the top two elements of the stack and replaces them with its result. When both an action and operator are specified in the same shortword, the action is performed followed by the operation.

The binary operator can also be from the set { ENF_COR, ENF_CAND, ENF_CNOR, ENF_CNAND }. These are "short-circuit" operators, in that they terminate the execution of the filter immediately if the condition they are checking for is found, and continue otherwise. All pop two elements from the stack and compare them for equality; ENF_CAND returns false if the result is false; ENF_COR returns true if the result is true; ENF_CNAND returns true if the result is false; ENF_CNOR returns false if the result is true. Unlike the other binary operators, these four do not leave a result on the stack, even if they continue.

The short-circuit operators should be used when possible, to reduce the amount of time spent evaluating filters. When they are used, you should also arrange the order of the tests so that the filter will succeed or fail as soon as possible; for example, checking the IP destination field of a UDP packet is more likely to indicate failure than the packet type field.

The special action ENF_NOPUSH and the special operator ENF_NOP can be used to only perform the binary operation or to only push a value on the stack. Since both are (conveniently) defined to be zero, indicating only an action actually specifies the action followed by ENF_NOP, and indicating only an operation actually specifies ENF_NOPUSH followed by the operation.

After executing the filter command list, a non-zero value (true) left on top of the stack (or an empty stack) causes the incoming packet to be accepted and a zero value (false) causes the packet to be rejected. (If the filter exits as the result of a short-circuit operator, the top-of-stack value is ignored.) Specifying an undefined operation or action in the command list or performing an illegal operation or action (such as pushing a shortword offset past the end of the packet or executing a binary operator with fewer than two shortwords on the stack) causes a filter to reject the packet.

EXAMPLES

The reverse ARP daemon program (**rarpd**(8C)) uses code similar to the following fragment to construct a filter that rejects all but RARP packets. That is, is accepts only packets whose Ethernet type field has the value ETHERTYPE_REVARP.

```
struct ether_header eh;            /* used only for offset values */
struct packetfilt pf;
register u_short *fwp = pf.Pf_Filter;
u_short offset;

/*
 * Set up filter. Offset is the displacement of the Ethernet
 * type field from the beginning of the packet in units of
 * u_shorts.
 */
```

```
offset = ((u_int) &eh.ether_type - (u_int) &eh.ether_dhost) / sizeof (u_short);
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT;
*fwp++ = htons(ETHERTYPE_REVARP);
*fwp++ = ENF_EQ;
pf.Pf_FilterLen = fwp - &pf.Pf_Filter[0];
```

This filter can be abbreviated by taking advantage of the ability to combine actions and operations:

```
...
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT | ENF_EQ;
*fwp++ = htons(ETHERTYPE_REVARP);
...
```

WARNINGS

The module name '**pf**' used in the system configuration file and as argument to the **I_PUSH** *ioctl* is provisional and subject to change.

The *Pf_Priority* field of the *packetfilt* structure is likely to be removed.

SEE ALSO

inet(4F), **nit**(4P), **nit_buf**(4M), **nit_if**(4M)

NAME

null – data sink

CONFIG

None; included with standard system.

SYNOPSIS

#include <fcntl.h>

open("/dev/null", *mode***);**

DESCRIPTION

Data written on the **null** special file is discarded.

Reads from the **null** special file always return an end-of-file indication.

FILES

/dev/null

NAME

openprom – PROM monitor configuration interface

CONFIG

pseudo-device openeepr

SYNOPSIS

#include <fcntl.h>
#include <sys/types.h>
#include <sundev/openpromio.h>
open("/dev/openprom", mode);

AVAILABILITY

SPARCstation 1 systems only.

DESCRIPTION

As with other Sun systems, configuration options are stored in an EEPROM or NVRAM on a SPARCstation 1 system. However, unlike other Sun systems, the encoding of these options is private to the PROM monitor. The **openprom** device provides an interface to the PROM monitor allowing a user program to query and set these configuration options through the use of **ioctl**(2) requests. These requests are defined in **<sundev/openpromio.h>**:

```
struct openpromio {
        u_int   oprom_size;             /* real size of following array */
        char    oprom_array[1];         /* For property names and values */
                                        /* NB: Adjacent, Null terminated */
};
#define OPROMMAXPARAM       1024        /* max size of array */

#define OPROMGETOPT         _IO(O,1)
#define OPROMSETOPT         _IO(O,2)
#define OPROMNXTOPT         _IO(O,3)
```

For all **ioctl**( ) requests, the third parameter is a pointer to a '**struct openpromio**'. All property names and values are null-terminated strings; the value of a numeric option is its ASCII representation.

IOCTLS

The **OPROMGETOPT** ioctl takes the null-terminated name of a property in the *oprom_array* and returns its null-terminated value (overlaying its name). *oprom_size* should be set to the size of *oprom_array*; on return it will contain the size of the returned value. If the named property does not exist, or if there is not enough space to hold its value, then *oprom_size* will be set to zero. See BUGS below.

The **OPROMSETOPT** ioctl takes two adjacent strings in *oprom_array*; the null-terminated property name followed by the null-terminated value.

The **OPROMNXTOPT** ioctl is used to retrieve properties sequentially. The null-terminated name of a property is placed into *oprom_array* and on return it is replaced with the null-terminated name of the next property in the sequence, with *oprom_size* set to its length. A null string on input means return the name of the first property; an *oprom_size* of zero on output means there are no more properties.

ERRORS

EINVAL	The size value was invalid, or (for **OPROMSETOPT**) the property does not exist.
ENOMEM	The kernel could not allocate space to copy the user's structure

FILES

/dev/openprom	PROM monitor configuration interface

SEE ALSO

mem(4S), **eeprom**(8S), **monitor**(8S)

BUGS

There should be separate return values for non-existent properties as opposed to not enough space for the value.

An attempt to set a property to an illegal value results in the PROM setting it to some legal value, with no error being returned. An **OPROMGETOPT** should be performed after an **OPROMSETOPT** to verify that the set worked.

The driver should be more consistent in its treatment of errors and edge conditions.

NAME
pp – Centronics-compatible parallel printer port

CONFIG — Sun386i SYSTEMS
device pp0 at obio ? csr 0x378 irq 15 priority 2

CONFIG — SUN-3x SYSTEMS
device pp0 at obio ? csr 0x6f000000 priority 1

This synopsis line should be used to generate a kernel for Sun-3/80 systems only.

AVAILABILITY
Sun386i and Sun-3/80 systems only.

DESCRIPTION
This device driver provides an interface to the Sun386i and Sun-3/80 systems' on-board Centronics-compatible parallel printer port. It supports most standard PC printers with Centronics interfaces.

FILES
/dev/pp0

DIAGNOSTICS
pp*: printer not online
pp*: printer out of paper

NAME

pty – pseudo-terminal driver

CONFIG

pseudo-device pty*n*

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/ttyp*n*", mode);
open("/dev/ptyp*n*", mode);

DESCRIPTION

The **pty** driver provides support for a pair of devices collectively known as a *pseudo-terminal*. The two devices comprising a pseudo-terminal are known as a *controller* and a *slave*. The slave device distinguishes between the **B0** baud rate and other baud rates specified in the **c_cflag** word of the **termios** structure, and the **CLOCAL** flag in that word. It does not support any of the other **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Instead of having a hardware interface and associated hardware that supports the terminal functions, the functions are implemented by another process manipulating the controller device of the pseudo-terminal.

The controller and the slave devices of the pseudo-terminal are tightly connected. Any data written on the controller device is given to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the controller device (rather than being transmitted from a UART).

In configuring, if no optional ''count'' is given in the specification, 16 pseudo-terminal pairs are configured.

IOCTLS

The standard set of **termio ioctls** are supported by the slave device. None of the bits in the **c_cflag** word have any effect on the pseudo-terminal, except that if the baud rate is set to **B0**, it will appear to the process on the controller device as if the last process on the slave device had closed the line; thus, setting the baud rate to **B0** has the effect of ''hanging up'' the pseudo-terminal, just as it has the effect of ''hanging up'' a real terminal.

There is no notion of ''parity'' on a pseudo-terminal, so none of the flags in the **c_iflag** word that control the processing of parity errors have any effect. Similarly, there is no notion of a ''break'', so none of the flags that control the processing of breaks, and none of the **ioctl**s that generate breaks, have any effect.

Input flow control is automatically performed; a process that attempts to write to the controller device will be blocked if too much unconsumed data is buffered on the slave device. The input flow control provided by the **IXOFF** flag in the **c_iflag** word is not supported.

The delays specified in the **c_oflag** word are not supported.

As there are no modems involved in a pseudo-terminal, the **ioctl**s that return or alter the state of modem control lines are silently ignored.

On Sun systems, an additional **ioctl** is provided:

TIOCCONS
The argument is ignored. All output that would normally be sent to the console (either from programs writing to **/dev/console** or from kernel printouts) is redirected so that it is written to the pseudo-terminal instead.

A few special **ioctls** are provided on the controller devices of pseudo-terminals to provide the functionality needed by applications programs to emulate real hardware interfaces:

TIOCSTOP
The argument is ignored. Output to the pseudo-terminal is suspended, as if a **STOP** character had been typed.

TIOCSTART
The argument is ignored. Output to the pseudo-terminal is restarted, as if a **START** character had been typed.

TIOCPKT
The argument is a pointer to an **int**. If the value of the **int** is non-zero, *packet* mode is enabled; if the value of the **int** is zero, packet mode is disabled. When a pseudo-terminal is in packet mode, each subsequent **read**(2V) from the controller device will return data written on the slave device preceded by a zero byte (symbolically defined as **TIOCPKT_DATA**), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD
whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
whenever the write queue for the terminal is flushed.

TIOCPKT_STOP whenever output to the terminal is stopped using ^S.

TIOCPKT_START whenever output to the terminal is restarted.

TIOCPKT_DOSTOP whenever XON/XOFF flow control is enabled after being disabled; it is considered ''enabled'' when the **IXON** flag in the **c_iflag** word is set, the **VSTOP** member of the **c_cc** array is ^S and the **VSTART** member of the **c_cc** array is ^Q.

TIOCPKT_NOSTOP whenever XON/XOFF flow control is disabled after being enabled.

This mode is used by **rlogin**(1C) and **rlogind**(8C) to implement a remote-echoed, locally ^S/^Q flow-controlled remote login with proper back-flushing of output when interrupts occur; it can be used by other similar programs.

TIOCREMOTE
The argument is a pointer to an **int**. If the value of the **int** is non-zero, *remote* mode is enabled; if the value of the **int** is zero, remote mode is disabled. This mode can be enabled or disabled independently of packet mode. When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and not input edited (regardless of the mode the slave side of the pseudo-terminal). Each write to the controller device produces a record boundary for the process reading the slave device. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an **EOF** character. Note: this means that a process writing to a pseudo-terminal controller in *remote* mode must keep track of line boundaries, and write only one line at a time to the controller. If, for example, it were to buffer up several **NEWLINE** characters and write them to the controller with one **write**( ), it would appear to a process reading from the slave as if a single line containing several **NEWLINE** characters had been typed (as if, for example, a user had typed the **LNEXT** character before typing all but the last of those **NEWLINE** characters). Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

The **ioctls TIOCGWINSZ**, **TIOCSWINSZ**, and, on Sun systems, **TIOCCONS**, can be performed on the controller device of a pseudo-terminal; they have the same effect as when performed on the slave device.

FILES

/dev/pty[p-s][0-9a-f]	pseudo-terminal controller devices
/dev/tty[p-s][0-9a-f]	pseudo-terminal slave devices
/dev/console	

SEE ALSO

rlogin(1C), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **rlogind**(8C)

BUGS

It is apparently not possible to send an **EOT** by writing zero bytes in **TIOCREMOTE** mode.

NAME

rfs, RFS – remote file sharing

CONFIGURATION

options RFS
options VFSSTATS

AVAILABILITY

Available only with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

The Remote File Sharing service, or RFS, allows transparent resource sharing among hosts on a network. A *resource* can be a directory, the files contained in that directory, subdirectories, devices, and even named pipes. Resources are advertised as a local directory using the name services. Hosts can then mount these resources, and use them as they would a local file system. The host advertising the resource is a file server, the hosts mounting the resource are clients.

All file servers and clients on a network belong to an RFS *domain*, and are administered by the same RFS name server. A domain consists of the following:

- A primary name server
- Possibly one or more secondary name servers
- File servers
- Clients

The name server maintains a list of advertised resources, and passwords in use. The name server also provides *name-to-resource* mapping. This allows a client to mount an advertised resource by the resource name, without needing to know the name of the file server or the pathname of the directory.

FILES

/usr/nserve/rfmaster hosts providing domain name service

SEE ALSO

clone(4), **nit_buf**(4M), **nit_pm**(4M), **tcptli**(4P), **timod**(4), **tirdwr**(4), **rfadmin**(8), **rfstart**(8), **rfudaemon**(8), **rmntstat**(8)

System and Network Administration

NAME

root – pseudo-driver for Sun386i root disk

CONFIG

pseudo-device rootdev

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **root** pseudo-driver provides indirect, device-independent access to the root disk on a diskful Sun workstation. The root disk is the disk where the mounted root partition resides - typically the disk from which the system was booted.

The intent of the **root** device is to allow uniform access to the partitions on the root disk, regardless of the disk's controller type or unit number. For example, the following version of **/etc/fstab** will work for any disk (assuming the disk has the standard partitions and filesystems):

```
/dev/roota /      4.2 rw 1 1
/dev/rootg /usr   4.2 ro 1 2
/dev/rooth /export 4.2 rw 1 3
```

When the root device is opened, the open and all subsequent operations on that device (**read**(2V), **write**(2V), **ioctl**(2), **close**(2V)) are redirected to the real disk. Therefore, all device-dependent operations on a particular disk are still accessible via the root device (see **dkio**(4S)).

FILES

/dev/root[a-h]	block partitions
/dev/rroot[a-h]	raw partitions

SEE ALSO

fstab(5), **sd**(4S), **open**(2V), **dkio**(4S)

NAME

routing – system supporting for local network packet routing

DESCRIPTION

The network facilities provided general packet routing, leaving routing table maintenance to applications processes.

A simple set of data structures comprise a ''routing table'' used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing daemon, maintains this data base with the aid of two socket specific **ioctl**(2) commands, **SIOCADDRT** and **SIOCDELRT**. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by super-user.

A routing table entry has the following form, as defined in **<net/route.h>**:

```
struct rtentry {
        u_long  rt_hash;
        struct  sockaddr rt_dst;
        struct  sockaddr rt_gateway;
        short   rt_flags;
        short   rt_refcnt;
        u_long  rt_use;
        struct  ifnet *rt_ifp;
};
```

with *rt_flags* defined from:

```
#define RTF_UP         0x1     /* route usable */
#define RTF_GATEWAY    0x2     /* destination is a gateway */
#define RTF_HOST       0x4     /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). When the system is booted, each network interface autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is a ''direct'' connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route.

User processes read the routing tables through the **/dev/kmem** device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

FILES

/dev/kmem

SEE ALSO

ioctl(2), **route**(8C), **routed**(8C)

NAME

sd – driver for SCSI disk devices

CONFIG — SUN-3, SUN-3x, and SUN-4 SYSTEMS

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si0 at obio ? csr 0x140000 priority 2
disk sd0 at si0 drive 0 flags 0
disk sd1 at si0 drive 1 flags 0
disk sd2 at si0 drive 8 flags 0
disk sd3 at si0 drive 9 flags 0
disk sd4 at si0 drive 16 flags 0
disk sd6 at si0 drive 24 flags 0

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd3 at sc0 drive 9 flags 0
disk sd4 at sc0 drive 16 flags 0
disk sd6 at sc0 drive 24 flags 0

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

The four lines following the **controller** specification lines define the available **disk** devices, **sd0** – **sd6**.

The **flags** field is used to specify the SCSI device type to the host adapter. **flags** must be set to 0 to identify **disk** devices.

The **drive** value is calculated using the formula:

$8 * target + lun$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the configuration for the older **sc0** host adapter. It uses the same configuration description as the **si0** host adapter.

CONFIG — SPARCsystem 330 and SUN-3/80 SYSTEMS

controller sm0 at obio ? csr 0xfa000000 priority 2
disk sd0 at sm0 drive 0 flags 0
disk sd1 at sm0 drive 1 flags 0
disk sd2 at sm0 drive 8 flags 0
disk sd3 at sm0 drive 9 flags 0
disk sd4 at sm0 drive 16 flags 0
disk sd6 at sm0 drive 24 flags 0

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**. It follows the same rules as described above for the Sun-3, Sun-3x, and Sun-4 section.

CONFIG — SUN-4/110 SYSTEM

controller sw0 at obio 2 csr 0xa000000 priority 2
disk sd0 at sw0 drive 0 flags 0
disk sd1 at sw0 drive 1 flags 0
disk sd2 at sw0 drive 8 flags 0
disk sd3 at sw0 drive 9 flags 0
disk sd4 at sw0 drive 16 flags 0
disk sd6 at sw0 drive 24 flags 0

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**. It follows the same rules as described above for the Sun-3, and Sun-4 section.

CONFIG — SUN-3/E SYSTEM

controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
disk sd0 at se0 drive 0 flags 0
disk sd1 at se0 drive 1 flags 0
disk sd2 at se0 drive 8 flags 0
disk sd3 at se0 drive 9 flags 0

The Sun-3/E uses a VME-based SCSI host adapter, **se0**. It follows the same rules as described above for the Sun-3 and Sun-4 section.

CONFIG — Sun386i

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
disk sd0 at wds0 drive 0 flags 0
disk sd1 at wds0 drive 8 flags 0
disk sd2 at wds0 drive 16 flags 0

The Sun386i configuration follows the same rules described above under the Sun-3 and Sun-4 configuration section. configuration section.

CONFIG — SPARCstation 1 SYSTEMS

device-driver esp
scsibus0 at esp
disk sd0 at scsibus0 target 3 lun 0
disk sd1 at scsibus0 target 1 lun 0
disk sd2 at scsibus0 target 2 lun 0
disk sd3 at scsibus0 target 0 lun 0

The SPARCstation 1 configuration files specify a device driver (**esp**), and a SCSI bus attached to that device driver, and then disks on that SCSI bus at the SCSI Target and Logical Unit addresses are specified.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0. The standard device names begin with "**sd**" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character **?** stands here for a drive number in the range 0-6.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the raw files conventionally begin with an extra '**r**.'

I/O requests (such as **lseek (2V)**) to the SCSI disk must have an offset that is a multiple of 512 bytes (DEV_BSIZE), or the driver returns an EINVAL error. If the transfer length is not a multiple of 512 bytes, the transfer count is rounded up by the driver.

Disk Support

This driver handles the Adaptec ACB-4000 disk controller for ST-506 drives, the Emulex MD21 disk controller for ESDI drives, and embedded, CCS-compatible SCSI disk drives.

On Sun386i and SPARCstation 1 systems, this driver supports the CDC Wren III half-height, and Wren IV full-height SCSI disk drives.

The type of disk drive is determined using the SCSI inquiry command and reading the volume label stored on block 0 of the drive. The volume label describes the disk geometry and partitioning; it must be present or the disk cannot be mounted by the system.

The **sd?a** partition is normally used for the root file system on a disk, the **sd?b** partition as a paging area (e.g. swap), and the **sd?c** partition for pack-pack copying. **sd?c** normally maps the entire disk and may also be used as the mount point for secondary disks in the system. The rest of the disk is normally the **sd?g** partition. For the primary disk, the user file system is located here.

FILES

/dev/sd[0-6][a-h]	block files
/dev/rsd[0-6][a-h]	raw files

SEE ALSO

dkio(4S), **directory**(3V), **lseek**(2V), **read**(2V), **write**(2V)

Product Specification for Wren IV SCSI Model 94171
Product Specification for Wren III SCSI Model 94161
Product Specification for Wren III SCSI Model 94211
Emulex MD21 Disk Controller Programmer Reference Manual
Adaptec ACB-4000 Disk Controller OEM Manual

DIAGNOSTICS

sd?: sdtimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

sd?: sdtimer: can't abort request
The driver is unable to find the request in the disconnect queue to notify the device driver that it has failed.

sd?: no space for inquiry data
sd?: no space for disk label
The driver was unable to get enough space for temporary storage. The driver is unable to open the disk device.

sd?: <%s>
The driver has found a SCSI disk device and opened it for the first time. The disk label is displayed to notify the user.

sd?: SCSI bus failure
A host adapter error was detected. The system may need to be rebooted.

sd?: single sector I/O failed
The driver attempted to recover from a transfer by writing each sector, one at a time, and failed. The disk needs to be reformatted to map out the new defect causing this error.

sd?: retry failed
sd?: rezero failed
A disk operation failed. The driver first tries to recover by retrying the command, if that fails, the driver rezeros the heads to cylinder 0 and repeats the retries. A failure of either the retry or rezero operations results in these warning messages; the error recovery operation continues until the retry count is exhausted. At that time a hard error is posted.

sd?: request sense failed
The driver was attempting to determine the cause of an I/O failure and was unable to get more information. This implies that the disk device may have failed.

sd?: warning, abs. block %d has failed %d times
The driver is warning the user that the specified block has failed repeatedly.

sd?: block %d needs mapping

sd?: reassigning defective abs. block %d

The specified block has failed repeatedly and may soon become an unrecoverable failure. If the driver does not map out the specified block automatically, it is recommend that the user correct the problem.

sd?: reassign block failed

The driver attempted to map out a block having excessive soft errors and failed. The user needs to run format and repair the disk.

sd?%c: cmd how blk %d (rel. blk %d)

sense key(0x%x): %s, error code(0x%x): %s

An I/O operation (**cmd**), encountered an error condition at absolute block (**blk %d**), partition (**sd?%c:**), or relative block (**rel. block%d**). The error recovery operation (how) indicates whether it *retry*'ed, *restored*, or *failed*. The **sense key** and **error code** of the error are displayed for diagnostic purposes. The absolute **blk** of the the error is used for mapping out the defective block. The **rel. blk** is the block (sector) in error, relative to the beginning of the partition involved. This is useful for using **icheck**(8) to repair a damaged file structure on the disk.

SPARCstation 1 Diagnostics

The diagnostics for SPARCstation 1 are much like as above. Below are some additional diagnostics you might see on a SPARCstation 1:

sd?: SCSI transport failed: reason 'xxxx': {retrying|giving up}

The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

sd?: disk not responding to selection

The target disk isn't responding. You may have accidently kicked a power cord loose.

sd?: disk ok

The target disk is now responding again.

sd?: disk offline

The driver has decided that the target disk is no longer there.

BUGS

These disk drivers assume that you don't have removable media drives, and also that in order to operate normally, a valid Sun disk label must be in sector zero.

A logical block size of 512 bytes is assumed (and enforced on SPARCstation 1).

NAME

sockio – ioctls that operate directly on sockets

SYNOPSIS

#include <sys/sockio.h>

DESCRIPTION

The IOCTL's listed in this manual page apply directly to sockets, independent of any underlying protocol. Note: the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. **ioctls** for a specific network interface or protocol are documented in the manual page for that interface or protocol.

SIOCSPGRP The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl** to the value of that **int**.

SIOCGPGRP The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl**.

SIOCCATMARK The argument is a pointer to an **int**. Set the value of that **int** to 1 if the read pointer for the socket referred to by the descriptor passed to **ioctl** points to a mark in the data stream for an out-of-band message, and to 0 if it does not point to a mark.

SEE ALSO

ioctl(2), **getsockopt**(2), **filio**(4)

NAME

sr – driver for CDROM SCSI controller

CONFIG — SPARCstation 1 and SPARCserver

disk sr0 at scsibus0 target 6 lun 0

CONFIG — SUN-4/330 SYSTEMS

disk sr0 at sm0 drive 060 flags 2

CONFIG — SUN-4 SYSTEMS

disk sr0 at sc0 drive 060 flags 2
disk sr0 at si0 drive 060 flags 2

AVAILABILITY

SPARCstation 1, SPARCserver 1, and Sun-4/330 systems only.

DESCRIPTION

CDROM is a removable read-only direct-access device connected to the system's SCSI bus. **CDROM** drives are designed to work with any disc that meets the Sony-Philips "red-book" or "yellow-book" documents. They can read **CDROM** data discs, digital audio discs (Audio CD's) or combined-mode discs (that is, some tracks are audio, some tracks are data). A **CDROM** disc is singled sided containing approximately 540 mega-bytes of data or 74 minutes of audio.

The **CDROM** drive controller is set up as SCSI target 6. There is only a single logically unit number 0. Therefore, the minor device number is always 0.

Since all the other SCSI target ids has been reserved by the system, the system only supports one **CDROM** drive. The device names are **/dev/sr0** for block device and **/dev/rsr0** for character device.

The device driver supports **open**(2V), **read**(2V), **close**(2V) function calls through its block device and character device interface. In addition, it supports ioctl function call through the character device interface. When the device is first opened, the **CDROM** drive's eject button will be disabled (which prevents the manual removal of the disc) until the last **close**(2V) is called.

CDROM Drive Support

This driver supports the SONY CDU-8012 **CDROM** drive controller and other **CDROM** drives which has the same SCSI command set as the SONY CDU-8012. The type of **CDROM** drive is determined using the SCSI inquiry command.

There is no volume label stored on the **CDROM**. The disc geometry and paritioning information is always the same. If the **CDROM** is in ISO 9660 or **High Sierra Disk** format, it can be mounted as a file system.

FILES

/dev/sr0	block files
/dev/rsr0	raw files

SEE ALSO

cdromio(4S), **fstab**(5), **mount**(8)

NAME

st – driver for SCSI tape devices

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller si0 at obio ? csr 0x140000 priority 2
tape st0 at si0 drive 32 flags 1
tape st1 at si0 drive 40 flags 1
tape st2 at si1 drive 32 flags 1
tape st3 at si1 drive 40 flags 1

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

Following the **controller** specification lines are four lines which define the available **tape** devices, **st0–st3**. The first two **tape** devices, **st0** and **st1**, are on the first **controller**, **si0**. The next two **tape** devices, **st2** and **st3**, are on the second **controller**, **si1**.

The **flags** field is used to specify the SCSI device type to the host adapter. The **flags** field must be set to 1 to identify **tape** devices.

The **drive** value is calculated using the formula:

$8 * target + lun$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the older **sc0** host adapter configuration. It follows the same configuration description as the **si0** host adapter.

CONFIG — SPARCsystem 330, SUN–3/80 SYSTEMS

controller sm0 at obio ? csr 0xfa000000 priority 2
tape st0 at sm0 drive 32 flags 1
tape st1 at sm0 drive 40 flags 1

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**, which follows the rules described above in the Sun-3, Sun-3x, and Sun-4 section.

CONFIG — SUN–4/110 SYSTEM

controller sw0 at obio 2 csr 0xa000000 priority 2
tape st0 at sw0 drive 32 flags 1
tape st1 at sw0 drive 40 flags 1

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**, which follows the rules described above in the Sun-3, Sun-3x, and Sun-4 section.

CONFIG — SUN–3/E SYSTEM

controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape st0 at se0 drive 32 flags 1
tape st1 at se0 drive 40 flags 1

The Sun-3/E uses a VME-based SCSI host adapter, **se0**, which follows the rules described above for Sun-3, Sun-3x, and Sun-4 systems.

CONFIG — Sun386i

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
tape st0 at wds0 drive 32 flags 1

The Sun386*i* configuration follows the rules described above in the Sun-3, Sun-3x, and Sun-4 configuration section.

CONFIG — SPARCstation 1 SYSTEM

device-driver esp
scsibus0 at esp
tape st0 at scsibus0 target 4 lun 0
tape st1 at scsibus0 target 5 lun 1

The SPARCstation 1 configuration files specify a device driver (**esp**), and a SCSI bus attached to that device driver, and then tapes on that SCSI bus at the SCSI Target and Logical Unit addresses are specified.

DESCRIPTION

The **st** device driver is an interface to various SCSI tape devices. Supported 1/4–inch cartridge devices include the Archive Viper QIC–150 streaming tape drive, the Emulex MT–02 tape controller, and the Sysgen SC4000 (except on SPARCstation 1) tape controller. **st** provides a standard interface to these various devices, see **mtio**(4) for details.

The driver can be opened with either rewind on close (**/dev/rst***) or no rewind on close (**/dev/nrst***) options. A maximum of four tape formats per device are supported (see FILES below). The tape format is specified using the device name. The four rewind on close formats for **st0**, for example, are **/dev/rst0**, **/dev/rst8**, **/dev/rst16**, and **/dev/rst24**.

Read Operation

Fixed-length I/O tape devices require the number of bytes read or written to be a multiple of the physical record size. For example, 1/4–inch cartridge tape devices only read or write multiples of 512 bytes.

Fixed-length tape devices read or write multiple records if the blocking factor is greater than 64512 bytes (minphys limit). These multiple writes are limited to 64512 bytes. For example, if a write request is issued for 65536 bytes using a 1/4–inch cartridge tape, two writes are issued; the first for 64512 bytes and the second for 1024 bytes.

Tape devices, which support variable-length I/O operations, such as 1/2–inch reel tape, may read or write a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver reads or writes multiple records to satisfy the request. These multiple records are limited to 65534 bytes. As an example, if a write request for 65540 bytes is issued using 1/2–inch reel tape, two records are written; one for 65534 bytes followed by one for 6 bytes.

If the driver is opened for reading in a different format than the tape is written in, the driver overrides the user selected format. For example, if a 1/4–inch cartridge tape is written in QIC-24 format and opened for reading in QIC–11, the driver will detect a read failure on the first read and automatically switch to QIC–24 to recover the data.

Note: If the **/dev/*st[0–3]** format is used, no indication is given that the driver has overridden the user selected format. Other formats issue a warning message to inform the user of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2–inch reel and QIC–150 tape drives for example).

If a file mark is encountered during reading, no error is reported but the number of bytes transferred is zero. The next read operation reads into the next file.

End of media is indicated by two successive zero transfer counts. No further reading should be performed past the end of recorded media.

If the read request size is 2048 bytes, the tape driver behaves as a disk device and honors seek positioning requests (see **lseek**(2)). If a file mark is crossed during a read operation, this function is disabled.

Write Operation

Writing is allowed at either the beginning of tape or after the last written file on the tape. Writing from the beginning of tape is performed in the user-specified format. The original tape format is used for appending onto previously written tapes. A warning message is issued if the driver has to override the user-specified format.

Care should be used when appending files onto 1/2–inch reel tape devices, since an extra file mark is appended after the last file to mark the end of recorded media. In other words, the last file on the tape ends with two file marks instead of one. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the end of recorded media **ioctl**( ) is provided to eliminate this problem by having the driver perform the positioning operation.

If the end of tape is encountered during writing, no error is reported but the number of bytes transferred is zero and no further writing is allowed. Trailer records may be written by first writing a file mark followed by the trailer records. It is important that these trailer records be kept as short as possible to prevent data loss.

Close Operation

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device name is used, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning **ioctl**( ), like rewind, is issued after writing, a file mark is written before repositioning the tape.

Note: For 1/2–inch reel tape devices, two file marks are written to mark the end of recorded media before rewinding or performing a file positioning **ioctl**( ).**Iftheuser**wrote mark before closing a 1/2–inch reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly.

If no data was written and the driver was opened for WRITE-ONLY access, a file mark is written thus creating a null file.

IOCTLS

The following ioctls are supported: forwardspace record, forwardspace file, backspace record, backspace file, backspace file mark, rewind, write file mark, offline, erase, retension, space to EOM, and get status.

The backspace file and forwardspace file tape operations are inverses. Thus, a forwardspace "–1" file is equivalent to a backspace "1" file. A backspace "0" file is the same as forwardspace "0" file; both position the tape device to the beginning of the current file.

Backspace file mark moves the tape backwards by file marks. The tape position will end on the beginning of tape side of the desired file mark. Devices which do not support this function, such as 1/4–inch cartridge tape, return an ENXIO error.

Backspace record and forwardspace record operations perform much like space file operations, except that they move by records instead of files. Variable-length I/O devices (1/2–inch reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4–inch cartridge tape, for example, spaces 512 byte physical records. The status ioctl residue count contains the number of files or records not skipped. Record skipping does not go past a file mark; file skipping does not go past the end of recorded media.

Spacing to the end of recorded media positions the tape at a location just after the last file written on the tape. For 1/4–inch cartridge tape, this is after the last file mark on the tape. For 1/2–inch reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.

The offline ioctl rewinds and, if appropriate, takes the device offline by unloading the tape. Tape must be inserted before the tape device can be used again.

The erase ioctl rewinds the tape, erases it completely, and returns to the beginning of tape.

The retension ioctl only applies to 1/4–inch cartridge tape devices. It is used to restore tape tension improving the tape's soft error rate after extensive start-stop operations or long-term storage. Devices which do not support this function, such as 1/2–inch reel tape, return an ENXIO error.

The get status ioctl call returns the drive id (mt_type), sense key error (mt_erreg), file number (mt_fileno), and record number (mt_blkno) of the last error. The residue count (mt_resid) is set to the number of bytes not transferred or files/records not spaced.

Note: The error status is reset by the get status ioctl call or the next read, write, or other ioctl operation. If no error has occurred (sense key is zero), the current file and record position are returned.

ERRORS

EACCES
: The driver is opened for write access and the tape is write protected, or an attempt is made to write on a write protected tape. For writing with QIC–150 tape drives, this error is also reported if the wrong tape media is used for writing.

EBUSY
: The tape device is already in use.

EIO
: During opening, the tape device is not ready because either no tape is in the drive, or the drive is not on-line. Once open, this error is returned if the requested I/O transfer could not be completed.

EINVAL
: The number of bytes read or written is not a multiple of the physical record size (fixed-length tape devices only).

ENXIO
: During opening, the tape device does not exist. On ioctl functions, this indicates that the tape device does not support the ioctl function.

FILES

For QIC–150 tape devices (Archive Viper):

/dev/rst[0–3]	QIC–150 Format
/dev/rst[8–11]	QIC–150 Format
/dev/rst[16–20]	QIC–150 Format
/dev/rst[24–28]	QIC–150 Format
/dev/nrst[0–3]	non-rewinding QIC–150 Format
/dev/nrst[8–11]	non-rewinding QIC–150 Format
/dev/nrst[16–19]	non-rewinding QIC–150 Format
/dev/nrst[24–27]	non-rewinding QIC–150 Format

For QIC–24 tape devices (Emulex MT–02 and Sysgen SC4000):

/dev/rst[0–3]	QIC–11 Format
/dev/rst[8–11]	QIC–24 Format
/dev/rst[16–20]	QIC–24 Format
/dev/rst[24–28]	QIC–24 Format
/dev/nrst[0–3]	non-rewinding QIC–11 Format
/dev/nrst[8–11]	non-rewinding QIC–24 Format
/dev/nrst[16–19]	non-rewinding QIC–24 Format
/dev/nrst[24–27]	non-rewinding QIC–24 Format

Note: The QIC–24 format is preferred over QIC–11 for Sun-3, Sun-3x, Sun-4, and Sun386*i* systems.

SEE ALSO

mt(1), **tar**(1), **mtio**(4), **dump**(8), **restore**(8)

Archive Viper QIC–150 Tape Drive Product Specification
Emulex MT–02 Intelligent Tape Controller Product Specification
Sysgen SC4000 Intelligent Tape Controller Product Specification

DIAGNOSTICS

st?: sttimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

st?: sttimer: can't abort request
The driver is unable to find the request in the disconnect que to notify the device driver that it has failed. A SCSI bus reset is issued to recover from this error.

st?: unknown SCSI device found
The SCSI device is not a tape device; it is some other type of SCSI device.

st?: warning, unknown tape drive found
The driver does not recognize the tape device. Only the default tape density is used; block size is set to the value specified by the tape drive.

st?: tape is write protected
The tape is write protected.

st?: wrong tape media for writing
For QIC–150 tape drives, this indicates that the user is trying to write on a DC–300XL (or equivalent) tape. Only DC–6150 (or equivalent) tapes can be used for writing.
Note: DC–6150 was formerly known as DC–600XTD.

st?: warning, rewinding tape
The driver is rewinding tape in order to set the tape format.

st?: warning, using alternate tape format
The driver is overriding the user-selected tape format and using the previously used format.

st?: warning, tape rewound
For Sysgen tape controllers, the tape may be rewound as a result of getting sense data.

st?: format change failed
The tape drive rejected the mode select command to change the tape format.

st?: file mark write failed
The driver was unable to write a file mark.

st?: warning, The tape may be wearing out or the head may need cleaning.
st?: read retries= %d, file= %d, block= %d
st?: write retries= %d, file= %d, block= %d
The number of allowable soft errors has been exceeded for this tape. Either the tape heads need cleaning or the tape is wearing out. If the tape is wearing out, continued usage of it is not recommended.

st?: illegal command
The SCSI command just issued was illegal. This message can result from issuing an inappropriate command, such as trying to write over previously written files on the tape. On foreign tape devices, this can also be caused by selecting the wrong tape format.

st?: error: sense key(0x%x): %s, error code(0x%x): %s
An error has occurred. The sense key message and error code are displayed for diagnostic purposes.

st?: stread: not modulo %d block size
st?: stwrite: not modulo %d block size
The read or write request size must be a multiple of the **%d** physical block size.

st?: file positioning error
st?: block positioning error
The driver was unable to position the tape to the desired file or block (record). This is probably caused by a damaged tape.

st?: SCSI transport failed: reason 'xxxx': {retrying|giving up}
The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up (SPARCstation 1) only.

BUGS

Foreign tape devices which do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for Sun supplied tape devices.

Foreign tape devices which do not report a blank check error at the end of recorded media cause file positioning operations to fail. Some tape drives for example, mistakenly report media error instead of blank check error.

"Cooked" mode for read and write operations is not supported.

Systems using the older **sc0** host adapter or the Sysgen SC4000 tape controller, prevent disk I/O over the SCSI bus while the tape is in use (during a rewind for example). This problem is caused by the fact that they do not support disconnect/reconnect to free the SCSI bus. Newer tape devices, like the the Emulex MT–02, and host adapters, like **si0**, eliminate this problem.

Some older systems may not support the QIC–24 format, and may complain (or exhibit erratic behavior) when the user attempts to use this format.

SPARCstation 1 does not support the Sysgen SC4000 tape controller, nor does it support 1/2" variable record length operations, record space operations, or implied seeking.

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fd, command, arg)
int fd, command;

DESCRIPTION

STREAMS (see **intro**(2)) ioctl commands are a subset of **ioctl**(2) commands that perform a variety of control functions on STREAMS. The arguments *command* and *arg* are passed to the file designated by *fd* and are interpreted by the *stream*head. Certain combinations of these arguments may be passed to a module or driver in the stream.

fd is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the stream referenced by *fd* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. Subsequent system calls will fail with *errno* set to this value.

IOCTLS

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH
Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the *stream*head. It then calls the open routine of the newly-pushed module.

I_PUSH will fail if one of the following occurs:

EINVAL	The module name is invalid.
EFAULT	*arg* points outside the allocated address space.
ENXIO	The open routine of the new module failed.
ENXIO	A hangup is received on the stream referred to by *fd*.

I_POP
Removes the module just below the *stream head* of the stream pointed to by *fd*. *arg* should be 0 in an **I_POP** request.

I_POP will fail if one of the following occurs:

EINVAL	No module is present on *stream*.
ENXIO	A hangup is received on the stream referred to by *fd*.

I_LOOK
Retrieves the name of the module just below the *stream head* of the stream pointed to by *fd*, and places it in a null-terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least **FMNAMESZ+1** bytes long. An '**#include <sys/conf.h>**' declaration is required.

I_LOOK will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.
EINVAL	No module is present on *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

I_FLUSH will fail if one of the following occurs:

EAGAIN	No buffers could be allocated for the flush message.
EINVAL	The value of *arg* is invalid.
ENXIO	A hangup is received on the stream referred to by *fd*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the **SIGPOLL** signal (see **sigvec**(2)) when a particular event has occurred on the stream associated with *fd*. **I_SETSIG** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the **SIGPOLL** signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S_HIPRI**.

Processes that wish to receive **SIGPOLL** signals must explicitly register to receive them using **I_SETSIG**. If several processes register to receive this signal for the same event on the same *stream*, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals.

I_SETSIG will fail if one of the following occurs:

EINVAL	The value of *arg* is invalid or *arg* is zero and the process is not registered to receive the **SIGPOLL** signal.
EAGAIN	A data structure could not be allocated to store the signal request.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above.

I_GETSIG will fail if one of the following occurs:

EINVAL — The process is not registered to receive the **SIGPOLL** signal.

EFAULT — *arg* points outside the allocated address space of the process.

I_FIND — This request compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present.

I_FIND will fail if one of the following occurs:

EFAULT — *arg* points outside the allocated address space of the process.

EINVAL — *arg* does not point to a valid module name.

I_PEEK — This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf    ctlbuf;
struct strbuf    databuf;
long             flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures (see **getmsg**(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to **RS_HIPRI, I_PEEK** will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the **RS_HIPRI** flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or **RS_HIPRI.**

I_PEEK will fail if one of the following occurs:

EFAULT — *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

I_SRDOPT — Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM — Byte-stream mode, the default.

RMSGD — Message-discard mode.

RMSGN — Message-nondiscard mode.

Read modes are described in **read**(2V).

I_SRDOPT will fail if one of the following occurs:

EINVAL — *arg* is not one of the above legal values.

I_GRDOPT — Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in **read**(2V).

I_GRDOPT will fail if one of the following occurs:

EFAULT — *arg* points outside the allocated address space of the process.

I_NREAD

Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue.

I_NREAD will fail if one of the following occurs:

EFAULT — *arg* points outside the allocated address space of the process.

I_FDINSERT

creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf    ctlbuf;
struct strbuf    databuf;
long             flags;
int              fd;
int              offset;
```

The *len* field in the *ctlbuf strbuf* structure (see **putmsg**(2)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other stream and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer to the *fd* stream's driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to **RS_HIPRI.** For non-priority messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I_FDINSERT** does not block on this condition. For non-priority messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

I_FDINSERT will fail if one of the following occurs:

EAGAIN — A non-priority message was specified, the **O_NDELAY** flag is set, and the stream write queue is full due to internal flow control conditions.

EAGAIN — Buffers could not be allocated for the message that was to be created.

EFAULT — *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

EINVAL	*fd* in the *strfdinsert* structure is not a valid, open stream file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly-aligned location in the data buffer; an undefined value is pointed to by *flags*.
ENXIO	A hangup is received on the stream referred to by *fd*.
ERANGE	The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to permit a process to specify timeouts and variable-sized amounts of data when sending an **ioctl** request to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY (see **open**(2V)) flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;        /* downstream command */
int     ic_timout;     /* ACK/NAK timeout */
int     ic_len;        /* length of data arg */
char    *ic_dp;        /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (−1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream.

I_STR will fail if one of the following occurs:

EAGAIN	Buffers could not be allocated for the **ioctl** message.

EFAULT — *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space of the process.

EINVAL — *ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than −1.

ENXIO — A hangup is received on the stream referred to by *fd*.

ETIME — A downstream **ioctl** timed out before acknowledgement was received.

An **I_STR** can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream*head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the **ioctl** command sent downstream fails. For these cases, **I_STR** will fail with *errno* set to the value in the message.

I_SENDFD Requests the stream associated with *fd* to send a message, containing a file pointer, to the *stream head* at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see **intro**(2)) of the *stream head* at the other end of the stream pipe to which it is connected.

I_SENDFD will fail if one of the following occurs:

EAGAIN — The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN — The read queue of the receiving *stream head* is full and cannot accept the message sent by **I_SENDFD**.

EBADF — *arg* is not a valid, open file descriptor.

EINVAL — *fd* is not connected to a stream pipe.

ENXIO — A hangup is received on the stream referred to by *fd*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD** **ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user ID and group ID, respectively, of the sending stream.

If **O_NDELAY** is not set (see **open**(2V)), **I_RECVFD** will block until a message is present at the *stream*head. If **O_NDELAY** is set, **I_RECVFD** will fail with *errno* set to EAGAIN if no message is present at the *stream*head.

If the message at the *stream head* is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*.

I_RECVFD will fail if one of the following occurs:

EAGAIN	A message was not present at the *stream head* read queue, and the **O_NDELAY** flag is set.
EBADMSG	The message at the *stream head* read queue was not a message containing a passed file descriptor.
EFAULT	*arg* points outside the allocated address space of the process.
EMFILE	Too many descriptors are active.
ENXIO	A hangup is received on the stream referred to by *fd*.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two streams, where *fd* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. **I_LINK** causes the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see **I_UNLINK**) on success, and a −1 on failure.

I_LINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fd*.
ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Storage could not be allocated to perform the **I_LINK**.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	The stream referred to by *fd* does not support multiplexing.
EINVAL	*arg* is not a stream, or is already linked under a multiplexor.
EINVAL	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fd*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_LINK** will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two streams specified by *fd* and *arg*. *fd* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the **ioctl I_LINK** command when a stream was linked below the multiplexing driver. If *arg* is −1, then all streams which were linked to *fd* are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink.

I_UNLINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fd*.

ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Buffers could not be allocated for the acknowledgement message.
EINVAL	The multiplexor ID number was invalid.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fd*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_UNLINK** will fail with *errno* set to the value in the message.

SEE ALSO

close(2V), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **open**(2V), **poll**(2), **putmsg**(2), **read**(2V), **sigvec**(2), **write**(2V)

STREAMS Programmer's Guide
STREAMS Primer

NAME

taac – Sun applications accelerator

CONFIG

taac0 at vme32d32 ? csr 0x28000000

CONFIG – SUN-3/SUN-4 SYSTEMS

device taac0 at vme32d32 1 csr 0x28000000
device taac0 at vme32d32 2 csr 0xf8000000
device taac0 at vme32d32 3 csr 0x28000000

The first line should be used to generate a kernel for Sun-3/160, Sun-3/260, Sun-4/260, Sun-4/370 and Sun-4/460 systems. The second line should be used to generate a kernel for Sun-4/110 systems; and the last line should be used to generate a kernel for Sun-4/330 systems.

CONFIG – SUN-4/150 SYSTEMS

device taac0 at vme32d32 2 csr 0xf8000000

AVAILABILITY

TAAC-1 can only be used in Sun VME-bus packages with 4 or more full size (9U) slots.

DESCRIPTION

The **taac** interface supports the optional TAAC-1 Applications Accelerator. This add-on device is composed of a very-long-instruction-word computation engine, coupled with an 8MB memory array. This memory area can be used as a frame buffer or as storage for large data sets.

the Sun-4/150 VME address space is limited to 28 bits. The TAAC-1 must be reconfigured to work in this package. See *Configuration Procedures fro the TAAC-1 Application Accelerator Board Set.*

Programs can be downloaded for execution on the TAAC-1 directly, they can be executed by the host processor, or the host processor and the TAAC-1 engine can be used in combination. See the *TAAC-1 User's Guide* for detailed information on accessing the TAAC-1 from the host. This manual also describes the C compiler, the programming tools, and the support libraries for the TAAC-1.

Programs on the host processor gain access to the TAAC-1 registers and memory by using **mmap**(2).

SEE ALSO

mmap(2)

TAAC-1 Application Accelerator: User Guide
Configuration Procedures for the TAAC-1 Application Accelerator Board Set

NAME

tcp – Internet Transmission Control Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol used to support the **SOCK_STREAM** abstraction. TCP is layered above the Internet Protocol (IP), the Internet protocol family's unreliable internetwork datagram delivery protocol.

TCP uses IP's host-level addressing and adds its own per-host collection of "port addresses". The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See **inet**(4F) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either "active" or "passive". Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the **bind**(2) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the **listen**(2) system call after binding the socket with **bind** . This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the **accept**(2) system call. Active sockets use the **connect**(2) call after binding to initiate connections.

By using the special value **INADDR_ANY**, the local IP address can be left unspecified in the **bind** call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the **read**(2V) and **write**(2V) system calls.

TCP supports one socket option which is set with **setsockopt** and tested with **getsockopt**(2). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (defined in **<netinet/tcp.h>**), to defeat this algorithm. The option level for the **setsockopt** call is the protocol number for TCP, available from **getprotobyname** (see **getprotoent**(3N)).

Options at the IP level may be used with TCP; see **ip**(4P).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of **send**(2). The caller may mark one byte as "urgent" with the **MSG_OOB** flag to **send**(2). This causes an "urgent pointer" pointing to this byte to be set in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a SIGURG signal. The **SIOCATMARK** ioctl returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single **read**(2V) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the **SIOCATMARK** ioctl, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

TCP assumes the datagram service it is layered above is unreliable. A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

ERRORS

A socket operation may fail if:

EISCONN	A **connect** operation was attempted on a socket on which a **connect** operation had already been performed.
ETIMEDOUT	A connection was dropped due to excessive retransmissions.
ECONNRESET	The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
ECONNREFUSED	The remote peer actively refused connection establishment (usually because no process is listening to the port).
EADDRINUSE	A **bind** operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A **bind** operation was attempted on a socket with a network address for which no network interface exists.
EACCES	A **bind** operation was attempted with a "reserved" port number and the effective user ID of the process was not super-user.
ENOBUFS	The system ran out of memory for internal data structures.

SEE ALSO

accept(2), **bind**(2), **connect**(2), **getsockopt**(2), **listen**(2), **read**(2V), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P)

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

BUGS

SIOCSHIWAT and **SIOCGHIWAT** ioctl's to set and get the high water mark for the socket queue, and so that it can be changed from 2048 bytes to be larger or smaller, have been defined (in **<sys/ioctl.h>**) but not implemented.

NAME

tcptli – TLI-Conforming TCP Stream-Head

CONFIG

pseudo-device clone

pseudo-device tcptli32

SYNOPSIS

#include <fcntl.h>
#include <nettli/tiuser.h>

tfd = t_open("/dev/tcp", O_RDWR, tinfo);
struct t_info *tinfo;

DESCRIPTION

TCPTLI provides access to TCP service via the Transport Library Interface (TLI). Prior to this release, TCP access was only possible via the socket programming interface. Programmers have the choice of using either the socket or TLI programming interface for their application.

TCPTLI is implemented in STREAMS conforming to the Transport Provider Interface (TPI) specification as a TCP Transport Provider to a TLI application. It utilizes the existing underlying socket and TCP support in the SunOS kernel to communicate over the network. It is also a clone driver, see **clone(4)** for more characteristics pertaining to a clone STREAMS driver.

The notion of an address is the same as the socket address (struct sockaddr_in) defined in <netinet/in.h>. TCPTLI maintains transport state information for each outstanding connection and the current state of the provider may be retrieved via the **t_getstate(3N)** call. See **t_getstate(3N)** for a list of possible states.

A server usually starts up with the **t_open(3N)** call followed by **t_bind(3N)** to bind an address that it listens for incoming connection. It may call **t_listen(3N)** to retrieve an indication of a connect request from another transport user, and then calls **t_accept(3N)** if it is willing to provide its service. TLI allows a server to accept connection on the same file descriptor it is listening on, or a different file descriptor (as in the sense of socket's **accept(2)**).

A client usually calls **t_open(3N)** and followed by a call to **t_bind(3N).** Then it calls **t_connect(3N)** to the address of a server advertized for providing service. Once the connection is established, it may use **t_rcv(3N)** and **t_snd(3N)** to receive and send data. The routine **t_close(3N)** is used to terminate the connection.

TLI ERRORS

An TLI operation may fail if one of the following error conditions is encountered. They are returned by the TLI user level library.

TBADADDR	Incorrect/invalid address format supplied by the user.
TBADOPT	Incorrect option.
TACCESS	No permission.
TBADF	Illegal transport file descriptor.
TNOADDR	Could not allocate address
TOUTSTATE	The transport is in an incorrect state.
TBADSEQ	Incorrect sequence number.
TSYSERR	A system error, i.e. below the transport level (see list below) is encountered.
TLOOK	An event requires attention.
TBADDATA	Illegal amount of data
TBUFOVFLW	Buffer not large enough.

TFLOW	Flow control problem.
TNODATA	No data.
TNODIS	No discon_ind is found on the queue.
TNOUDERR	Unit data not found.
TBADFLAG	Bad flags.
TNOREL	No orderly release request found on queue.
TNOTSUPPORT	Protocol/primitive is not supported.
TSTATECHNG	State is in the process of changing.

SYSTEM ERRORS

The following errors are returned by TCPTLI. However they may be translated to the above TLI errors by the user level library (**libnsl**).

ENXIO	Invalid device or address, out of range.
EBUSY	Request device is busy or not ready.
ENOMEM	Not enough memory for transmitting data, non fatal.
EPROTO	The operation encountered an underlying protocol. error (TCP).
EWOULDBLOCK	The operation would block as normally the file descriptors are set with non-blocking flag.
EACCES	Permission denied.
ENOBUFS	The system ran out of memory for internal (network) data structures.

SEE ALSO

accept(2), **t_open**(3N), **t_close**(3N), **t_accept**(3N), **t_getstate**(3N), **t_bind**(3N), **t_connect**(3N), **t_rcv**(3N), **t_snd**(3N), **t_alloc**(3N), **t_unbind**(3N), **t_getinfo**(3N)

BUGS

Only TCP (i.e. connection oriented) protocol is supported, no UDP. The maximum network connection is 32 by default. A new kernel has to be configured if an increase of such limit is desired: by changing the entry **pseudo-device tcptli32** in the kernel config file to **tcptli64.**

NAME

termio – general terminal interface

SYNOPSIS

#include <sys/termios.h>

DESCRIPTION

Asynchronous communications ports, pseudo-terminals, and the special interface accessed by **/dev/tty** all use the same general interface, no matter what hardware (if any) is involved. The remainder of this section discusses the common features of this interface.

Opening a Terminal Device File

When a terminal file is opened, the process normally waits until a connection is established. In practice, users' programs seldom open these files; they are opened by **getty**(8) and become a user's standard input, output, and error files. The state of the software carrier flag will effect the ability to open a line.

Sessions

Processes are now grouped by session, then process group, then process id. Each session is associated with one "login" session (windows count as logins). A process creates a session by calling **setsid**(2V), which will put the process in a new session as its only member and as the session leader of that session.

Process Groups

A terminal may have a distinguished process group associated with it. This distinguished process group plays a special role in handling signal-generating input characters, as discussed below in the **Special Characters** section below. The terminal's process group can can be set only to process groups that are members of the terminal's session.

A command interpreter, such as **csh**(1), that supports "job control" can allocate the terminal to different *jobs*, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's associated process group may be set or examined by a process with sufficient privileges. The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see **Job Access Control** below.

Orphaned Process Groups

An orphaned process group is a process group that has no parent, in a different process group, and in the same session. In other words, there is no process that can handle job control signals for the process group.

The Controlling Terminal

A terminal may belong to a process as its *controlling terminal*. If a process that is a session leader, and that does not have a controlling terminal, opens a terminal file not already associated with a session, the terminal associated with that terminal file becomes the controlling terminal for that process, and the terminal's distinguished process group is set to the process group of that process. (Currently, this also happens if a process that does not have a controlling terminal and is not a member of a process group opens a terminal. In this case, if the terminal is not associated with a session, a new session is created with a process group ID equal to the process ID of the process in question, and the terminal is assigned to that session. The process is made a member of the terminal's process group.)

If a process does not wish to acquire the terminal as a controlling terminal (as is the case with many daemons that open **/dev/console**), the process should or **O_NOCTTY** into the second argument to **open**(2V).

The controlling terminal is inherited by a child process during a **fork**(2V). A process relinquishes its control terminal when it changes its process group using **setsid**(2V), when it trys to change back to process group 0 via a **setpgrp**(2V) with arguments **(mypid, 0)**, or when it issues a **TIOCNOTTY ioctl**(2) call on a file descriptor created by opening the file **/dev/tty**. Both of the last two cases cause a **setsid**(2V) to be called on the process' behalf. This is an attempt to allow old binaries (that couldn't have known about **setsid**(2V)) to still acquire controlling terminals. It doesn't always work, see **setsid**(8V) for a workaround for those cases.

When a session leader that has a controlling terminal terminates, the distinguished process group of the controlling terminal is set to zero (indicating no distinguished process group). This allows the terminal to be acquired as a controlling terminal by a new session leader.

Closing a Terminal Device File

When a terminal device file is closed, the process closing the file waits until all output is drained; all pending input is then flushed, and finally a disconnect is performed. If **HUPCL** is set, the existing connection is severed (by hanging up the phone line, if appropriate).

Job Access Control

If a process is in the (non-zero) distinguished process group of its controlling terminal (if this is true, the process is said to be a *foreground process*), then **read**(2V) operations are allowed as described below in **Input Processing and Reading Characters**. If a process is not in the (non-zero) distinguished process group of its controlling terminal (if this is true, the process is said to be a *background process*), then any attempts to read from that terminal will typically send that process' process group a **SIGTTIN** signal. If the process is ignoring **SIGTTIN**, has **SIGTTIN** blocked, is a member of an orphaned process group, or is in the middle of process creation using **vfork**(2), the read will return −1 and set **errno** to **EIO**, and the **SIGTTIN** signal will not be sent. The **SIGTTIN** signal will normally stop the members of that process group.

When the **TOSTOP** bit is set in the **c_lflag** field, attempts by a background process to write to its controlling terminal will typically send that process' process group a **SIGTTOU** signal. If the process is ignoring **SIGTTOU**, has **SIGTTOU** blocked, or is in the middle of process creation using **vfork**( ), the process will be allowed to write to the terminal and the **SIGTTOU** signal will not be sent. If the process is orphaned, the write will return −1 and set **errno** to **EIO**, and the **SIGTTOU** signal will not be sent. **SIGTTOU** signal will normally stop the members of that process group. Certain **ioctl**() calls that set terminal parameters are treated in this same fashion, except that **TOSTOP** is not checked; the effect is identical to that of terminal writes when **TOSTOP** is set. See **IOCTLS**.

Input Processing and Reading Characters

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. This limit is available is {MAX_CANON} characters (see **pathconf**(2V)). If the **IMAXBEL** mode has not been selected, all the saved characters are thrown away without notice when the input limit is reached; if the **IMAXBEL** mode has been selected, the driver refuses to accept any further input, and echoes a bell (ASCII BEL).

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or non-canonical mode (see **ICANON** in the **Local Modes** section).

The style of input processing can also be very different when the terminal is put in non-blocking I/O mode; see **read**(2V). In this case, reads from the terminal will never block.

It is possible to simulate terminal input using the **TIOCSTI** **ioctl**( ) call, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the process' controlling terminal unless the process' effective user ID is super-user.

Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a NEWLINE (ASCII LF) character, an EOF (by default, an ASCII EOT) character, or one of two user-specified end-of-line characters, **EOL** and **EOL2**. This means that a **read**( ) will not complete until an entire line has been typed or a signal has been received. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

Erase and kill processing occurs during input. The **ERASE** character (by default, the character DEL) erases the last character typed in the current input line. The **WERASE** character (by default, the character CTRL-W) erases the last "word" typed in the current input line (but not any preceding SPACE or TAB characters). A "word" is defined as a sequence of non-blank characters, with TAB characters counted as blanks.

Neither **ERASE** nor **WERASE** will erase beyond the beginning of the line. The **KILL** character (by default, the character CTRL-U) kills (deletes) the entire current input line, and optionally outputs a NEWLINE character. All these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done.

The **REPRINT** character (the character CTRL-R) prints a NEWLINE followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; as a consequence, if **ECHO** is not set, they are not printed.

The **ERASE** and **KILL** characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The **ERASE** and **KILL** characters may be changed.

Non-Canonical Mode Input Processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The **MIN** and **TIME** values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (when the characters are returned to the user). **TIME** is a timer of 0.10 second granularity that is used to timeout bursty and short term data transmissions. The four possible values for **MIN** and **TIME** and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case **TIME** serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between **MIN** and **TIME** is as follows: as soon as one character is received, the intercharacter timer is started. If **MIN** characters are received before the intercharacter timer expires (remember that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before **MIN** characters are received, the characters received to that point are returned to the user. Note: if **MIN** expires at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (**MIN** > 0, **TIME** > 0) the read will sleep until the **MIN** and **TIME** mechanisms are activated by the receipt of the first character.

Case B: MIN > 0, TIME = 0

In this case, since the value of **TIME** is zero, the timer plays no role and only **MIN** is significant. A pending read is not satisfied until **MIN** characters are received (the pending read will sleep until **MIN** characters are received). A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since **MIN** = 0, **TIME** no longer represents an intercharacter timer. It now serves as a read timer that is activated as soon as a **read()** is done. A read is satisfied as soon as a single character is received or the read timer expires. Note: in this case if the timer expires, no character will be returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case the read will not block indefinitely waiting for a character – if no character is received within **TIME***.10 seconds after the read is initiated, the read will return with zero characters.

Case D: MIN = 0, TIME = 0

In this case return is immediate. The minimum of either the number of characters requested or the number of characters currently available will be returned without waiting for more characters to be input.

Comparison of the Different Cases of MIN, TIME Interaction

Some points to note about **MIN** and **TIME**:

- In the following explanations one may notice that the interactions of **MIN** and **TIME** are not symmetric. For example, when **MIN** > 0 and **TIME** = 0, **TIME** has no effect. However, in the opposite case where **MIN** = 0 and **TIME** > 0, both **MIN** and **TIME** play a role in that **MIN** is satisfied with the receipt of a single character.

- Also note that in case A (**MIN** > 0, **TIME** > 0), **TIME** represents an intercharacter timer while in case C (**TIME** = 0, **TIME** > 0) **TIME** represents a read timer.

These two points highlight the dual purpose of the **MIN/TIME** feature. Cases A and B, where **MIN** > 0, exist to handle burst mode activity (for example, file transfer programs) where a program would like to process at least **MIN** characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; while in case B, it is turned off.

Cases C and D exist to handle single character timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C the read is timed; while in case D, it is not.

Another important note is that **MIN** is always just a minimum. It does not denote a record length. That is, if a program does a read of 20 bytes, **MIN** is 10, and 25 characters are present, 20 characters will be returned to the user.

Writing Characters

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Special Characters

Certain characters have special functions on input and/or output. These functions and their default character values are summarized as follows:

INTR (CTRL-C or ASCII ETX) generates a **SIGINT** signal, which is sent to all processes in the distinguished process group associated with the terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see **sigvec**(2).

QUIT (CTRL-\ or ASCII FS) generates a **SIGQUIT** signal, which is sent to all processes in the distinguished process group associated with the terminal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

ERASE (Rubout or ASCII DEL) erases the preceding character. It will not erase beyond the start of a line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

WERASE (CTRL-W or ASCII ETB) erases the preceding "word". It will not erase beyond the start of a line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

KILL (CTRL-U or ASCII NAK) deletes the entire line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

REPRINT (CTRL-R or ASCII DC2) reprints all characters that have not been read, preceded by a NEWLINE.

EOF (CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a NEWLINE, and the **EOF** is discarded. Thus, if there are no characters waiting, which is to say the **EOF** occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed; it can, however, be escaped by the **LNEXT** character.

EOL
EOL2 (ASCII NUL) are additional line delimiters, like **NL**. They are not normally used.

SUSP (CTRL-Z or ASCII EM) is used by the job control facility to change the current job to return to the controlling job. It generates a **SIGTSTP** signal, which stops all processes in the terminal's process group.

STOP (CTRL-S or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, **STOP** characters are ignored and not read.

START (CTRL-Q or ASCII DC1) is used to resume output that has been suspended by a **STOP** character. While output is not suspended, **START** characters are ignored and not read.

DISCARD (CTRL-O or ASCII SI) causes subsequent output to be discarded until another **DISCARD** character is typed, more input arrives, or the condition is cleared by a program.

LNEXT (CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored; this works for all the special characters mentioned above. This allows characters to be input that would otherwise get interpreted by the system (for example, **KILL**, **QUIT**.)

The character values for **INTR**, **QUIT**, **ERASE**, **WERASE**, **KILL**, **REPRINT**, **EOF**, **EOL**, **EOL2**, **SUSP**, **STOP**, **START**, **DISCARD**, and **LNEXT** may be changed to suit individual tastes. If the value of a special control character is 0, the function of that special control character will be disabled. The **ERASE**, **KILL**, and **EOF** characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the **LNEXT** character, in which case no special function is done.

If **IEXTEN** is added to the local modes (this is the default), then all of the special characters are in effect. If **IEXTEN** is cleared from the local modes, then only the following POSIX.1 compatible specials are seen as specials: **INTR**, **QUIT**, **ERASE**, **KILL**, **EOF**, **NL**, **EOL**, **SUSP**, **STOP**, **START**, and **CR**.

Software Carrier Mode

The software carrier mode can be enabled or disabled using the **TIOCSSOFTCAR ioctl()**. If the software carrier flag for a line is off, the line pays attention to the hardware carrier detect (DCD) signal. The **tty** device associated with the line can not be opened until DCD is asserted. If the software carrier flag is on, the line behaves as if DCD is always asserted.

The software carrier flag is usually turned on for locally connected terminals or other devices, and is off for lines with modems.

To be able to issue the **TIOCGSOFTCAR** and **TIOCSSOFTCAR ioctl()** calls, the **tty** line should be opened with **O_NDELAY** so that the **open**(2V) will not wait for the carrier.

Modem Disconnect

If a modem disconnect is detected, and the **CLOCAL** flag is not set in the **c_cflag** field, a **SIGHUP** signal is sent to all processes in the distinguished process group associated with this terminal. Unless other arrangements have been made, this signal terminates the processes. If **SIGHUP** is ignored or caught, any subsequent **read()** returns with an end-of-file indication until the terminal is closed. Thus, programs that read a terminal and test for end-of-file can terminate appropriately after a disconnect. Any subsequent **write()** will return −1 and set **errno** to **EIO** until the terminal is closed.

A **SIGHUP** signal is sent to the **tty** if the software carrier flag is off and the hardware carrier detect drops.

Terminal Parameters

The parameters that control the behavior of devices and modules providing the **termios** interface are specified by the **termios** structure, defined by **<sys/termios.h>**. Several **ioctl()** system calls that fetch or change these parameters use this structure:

```
#define  NCCS      17
struct   termios {
         unsigned  long   c_iflag;     /* input modes */
         unsigned  long   c_oflag;     /* output modes */
         unsigned  long   c_cflag;     /* control modes */
```

```
        unsigned  long   c_lflag;      /* local modes */
        unsigned  char   c_line;       /* line discipline */
        unsigned  char   c_cc[NCCS];   /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions and initial values for each function are as follows:

0	VINTR	ETX
1	VQUIT	FS
2	VERASE	DEL
3	VKILL	NAK
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	VSWTCH	NUL
8	VSTART	DC1
9	VSTOP	DC3
10	VSUSP	EM
12	VREPRINT	DC2
13	VDISCARD	SI
14	VWERASE	ETB
15	VLNEXT	SYN

The **MIN** value is stored in the **VMIN** element of the **c_cc** array, and the **TIME** value is stored in the **VTIME** element of the **c_cc** array. The **VMIN** element is the same element as the **VEOF** element, and the **VTIME** element is the same element as the **VEOL** element.

Input Modes

The **c_iflag** field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.
IMAXBEL	0020000	Echo BEL on input line too long.

If **IGNBRK** is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if **BRKINT** is set, a break condition will generate a **SIGINT** and flush both the input and output queues. If neither **IGNBRK** nor **BRKINT** is set, a break condition is read as a single ASCII NUL character ('\0').

If **IGNPAR** is set, characters with framing or parity errors (other than break) are ignored. Otherwise, if **PARMRK** is set, a character with a framing or parity error that is not ignored is read as the three-character sequence: '\377', '\0', *X*, where *X* is the data of the character received in error. To avoid ambiguity in this case, if **ISTRIP** is not set, a valid character of '\377' is read as '\377', '\377'. If neither **IGNPAR** nor **PARMRK** is set, a framing or parity error (other than break) is read as a single ASCII NUL character ('\0').

If **INPCK** is set, input parity checking is enabled. If **INPCK** is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If **ISTRIP** is set, valid input characters are first stripped to 7 bits, otherwise all 8 bits are processed.

If **INLCR** is set, a received NL character is translated into a CR character. If **IGNCR** is set, a received CR character is ignored (not read). Otherwise if **ICRNL** is set, a received CR character is translated into a NL character.

If **IUCLC** is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If **IXON** is set, start/stop output control is enabled. A received **STOP** character will suspend output and a received **START** character will restart output. The **STOP** and **START** characters will not be read, but will merely perform flow control functions. If **IXANY** is set, any input character will restart output that has been suspended.

If **IXOFF** is set, the system will transmit a **STOP** character when the input queue is nearly full, and a **START** character when enough input has been read that the input queue is nearly empty again.

If **IMAXBEL** is set, the ASCII BEL character is echoed if the input stream overflows. Further input will not be stored, but any input already present in the input stream will not be disturbed. If **IMAXBEL** is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

The initial input control value is **BRKINT, ICRNL, IXON, ISTRIP**.

Output modes

The **c_oflag** field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	or tab expansion:
TAB1	0004000	
TAB2	0010000	
XTABS	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	

FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If **OPOST** is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If **OLCUC** is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with **IUCLC**.

If **ONLCR** is set, the NL character is transmitted as the CR-NL character pair. If **OCRNL** is set, the CR character is transmitted as the NL character. If **ONOCR** is set, no CR character is transmitted when at column 0 (first position). If **ONLRET** is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If **OFILL** is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If **OFDEL** is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If **ONLRET** is set, the RETURN delays are used instead of the NEWLINE delays. If **OFILL** is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If **OFILL** is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3, specified by **TAB3** or **XTABS**, specifies that TAB characters are to be expanded into SPACE characters. If **OFILL** is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If **OFILL** is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is **OPOST**, **ONLCR**, **XTABS**.

The **c_cflag** field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
B38400	0000017	38400 baud

CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
CIBAUD	03600000	Input baud rate, if different from output rate.
CRTSCTS	020000000000	Enable RTS/CTS flow control.

The **CBAUD** bits specify the baud rate. The zero baud rate, **B0**, is used to hang up the connection. If **B0** is specified, the modem control lines will cease to be asserted. Normally, this will disconnect the line. If the **CIBAUD** bits are not zero, they specify the input baud rate, with the **CBAUD** bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the **CBAUD** bits. The values for the **CIBAUD** bits are the same as the values for the **CBAUD** bits, shifted left **IBSHIFT** bits. For any particular hardware, impossible speed changes are ignored.

The **CSIZE** bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If **CSTOPB** is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are required.

If **PARENB** is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set, otherwise even parity is used.

If **CREAD** is set, the receiver is enabled. Otherwise no characters will be received.

If **HUPCL** is set, the modem control lines for the port will be disconnected when the last process with the line open closes it or terminates.

If **CLOCAL** is set, a connection does not depend on the state of the modem status lines. Otherwise modem control is assumed.

If **CRTSCTS** is set, and the terminal has modem control lines associated with it, the Request To Send (RTS) modem control line will be raised, and output will occur only if the Clear To Send (CTS) modem status line is raised. If the CTS modem status line is lowered, output is suspended until CTS is raised. Some hardware may not support this function, and other hardware may not permit it to be disabled; in either of these cases, the state of the **CRTSCTS** flag is ignored.

The initial hardware control value after open is **B9600**, **CS7**, **CREAD**, **PARENB**.

Local Modes

The **c_lflag** field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.
TOSTOP	0000400	Send **SIGTTOU** for background output.
ECHOCTL	0001000	Echo control characters as ^*char*, delete as ^?.
ECHOPRT	0002000	Echo erase character as character erased.
ECHOKE	0004000	BS-SP-BS erase entire line on line kill.

FLUSHO	0020000	Output is being flushed.
PENDIN	0040000	Retype pending input at next read or input character.
IEXTEN	0100000	Recognize all specials (if clear, POSIX only).

If **ISIG** is set, each input character is checked against the special control characters **INTR**, **QUIT**, and **SUSP**. If an input character matches one of these control characters, the function associated with that character is performed. If **ISIG** is not set, no checking is done. Thus these special input functions are possible only if **ISIG** is set.

If **ICANON** is set, canonical processing is enabled. This is affected by the **IEXTEN** bit (see **Special Characters** above). This enables the erase, word erase, kill, and reprint edit functions, and the assembly of input characters into lines delimited by NL, **EOF**, **EOL**, and **EOL2**. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds. See the *Non-canonical Mode Input Processing* section for more details.

If **XCASE** is set, and if **ICANON** is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for: *use*:
`   \'
| \!
~   \^
{   \(
}   \)
\   \\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If **ECHO** is set, characters are echoed as received. If **ECHO** is not set, input characters are not echoed.

If **ECHOCTL** is not set, all control characters (characters with codes between 0 and 37 octal) are echoed as themselves. If **ECHOCTL** is set, all control characters other than ASCII TAB, ASCII NL, the **START** character, and the **STOP** character, are echoed as ^*X*, where *X* is the character given by adding 100 octal to the control character's code (so that the character with octal code 1 is echoed as '**^A**'), and the ASCII DEL character, with code 177 octal, is echoed as '**^?**'.

When **ICANON** is set, the following echo functions are possible:

- If **ECHO** and **ECHOE** are set, and **ECHOPRT** is not set, the **ERASE** and **WERASE** characters are echoed as one or more ASCII BS SP BS, which will clear the last character(s) from a CRT screen.
- If **ECHO** and **ECHOPRT** are set, the first **ERASE** and **WERASE** character in a sequence echoes as a backslash (\) followed by the characters being erased. Subsequent **ERASE** and **WERASE** characters echo the characters being erased, in reverse order. The next non-erase character types a slash (/) before it is echoed.
- If **ECHOKE** is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by **ECHOE** and **ECHOPRT**).
- If **ECHOK** is set, and **ECHOKE** is not set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note: an escape character (\) or an **LNEXT** character preceding the erase or kill character removes any special function.
- If **ECHONL** is set, the NL character will be echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so-called half duplex).

- If **ECHOCTL** is not set, the **EOF** character is not echoed, unless it is escaped. Because EOT is the default **EOF** character, this prevents terminals that respond to EOT from hanging up. If **ECHOCTL** is set, the **EOF** character is echoed; if it is not escaped, after it is echoed, one backspace character is output if it is echoed as itself, and two backspace characters are echoed if it is echoed as ^X.

If **NOFLSH** is set, the normal flush of the input and output queues associated with the **INTR**, **QUIT**, and **SUSP** characters will not be done.

If **TOSTOP** is set, the signal **SIGTTOU** is sent to a process that tries to write to its controlling terminal if it is not in the distinguished process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring **SIGTTOU** signals are excepted and allowed to produce output.

If **FLUSHO** is set, data written to the terminal will be discarded. This bit is set when the **FLUSH** character is typed. A program can cancel the effect of typing the **FLUSH** character by clearing **FLUSHO**.

If **PENDIN** is set, any input that has not yet been read will be reprinted when the next character arrives as input.

The initial line-discipline control value is **ISIG, ICANON, ECHO**.

Minimum and Timeout

The **MIN** and **TIME** values are described above under **Non-canonical Mode Input Processing**. The initial value of **MIN** is 1, and the initial value of **TIME** is 0.

Termio Structure

The System V **termio** structure is used by other **ioctl**( ) calls; it is defined by **<sys/termio.h>** as:

```
#define  NCC       8
struct   termio {
         unsigned  short  c_iflag;     /* input modes */
         unsigned  short  c_oflag;     /* output modes */
         unsigned  short  c_cflag;     /* control modes */
         unsigned  short  c_lflag;     /* local modes */
         char             c_line;      /* line discipline */
         unsigned  char   c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions for each function are as follows:

0	**VINTR**
1	**VQUIT**
2	**VERASE**
3	**VKILL**
4	**VEOF**
5	**VEOL**
6	**VEOL2**
7	reserved

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

Terminal Size

The number of lines and columns on the terminal's display (or page, in the case of printing terminals) is specified in the **winsize** structure, defined by **<sys/termios.h>**. Several **ioctl**( ) system calls that fetch or change these parameters use this structure:

```
struct winsize {
         unsigned short    ws_row;     /* rows, in characters */
         unsigned short    ws_col;     /* columns, in characters */
```

FLUSHO	0020000	Output is being flushed.
PENDIN	0040000	Retype pending input at next read or input character.
IEXTEN	0100000	Recognize all specials (if clear, POSIX only).

If **ISIG** is set, each input character is checked against the special control characters **INTR**, **QUIT**, and **SUSP**. If an input character matches one of these control characters, the function associated with that character is performed. If **ISIG** is not set, no checking is done. Thus these special input functions are possible only if **ISIG** is set.

If **ICANON** is set, canonical processing is enabled. This is affected by the **IEXTEN** bit (see **Special Characters** above). This enables the erase, word erase, kill, and reprint edit functions, and the assembly of input characters into lines delimited by NL, **EOF**, **EOL**, and **EOL2**. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds. See the *Non-canonical Mode Input Processing* section for more details.

If **XCASE** is set, and if **ICANON** is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If **ECHO** is set, characters are echoed as received. If **ECHO** is not set, input characters are not echoed.

If **ECHOCTL** is not set, all control characters (characters with codes between 0 and 37 octal) are echoed as themselves. If **ECHOCTL** is set, all control characters other than ASCII TAB, ASCII NL, the **START** character, and the **STOP** character, are echoed as ^*X*, where *X* is the character given by adding 100 octal to the control character's code (so that the character with octal code 1 is echoed as '**^A**'), and the ASCII DEL character, with code 177 octal, is echoed as '**^?**'.

When **ICANON** is set, the following echo functions are possible:

- If **ECHO** and **ECHOE** are set, and **ECHOPRT** is not set, the **ERASE** and **WERASE** characters are echoed as one or more ASCII BS SP BS, which will clear the last character(s) from a CRT screen.
- If **ECHO** and **ECHOPRT** are set, the first **ERASE** and **WERASE** character in a sequence echoes as a backslash (\) followed by the characters being erased. Subsequent **ERASE** and **WERASE** characters echo the characters being erased, in reverse order. The next non-erase character types a slash (/) before it is echoed.
- If **ECHOKE** is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by **ECHOE** and **ECHOPRT**).
- If **ECHOK** is set, and **ECHOKE** is not set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note: an escape character (\) or an **LNEXT** character preceding the erase or kill character removes any special function.
- If **ECHONL** is set, the NL character will be echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so-called half duplex).

- If **ECHOCTL** is not set, the **EOF** character is not echoed, unless it is escaped. Because EOT is the default **EOF** character, this prevents terminals that respond to EOT from hanging up. If **ECHOCTL** is set, the **EOF** character is echoed; if it is not escaped, after it is echoed, one backspace character is output if it is echoed as itself, and two backspace characters are echoed if it is echoed as ^*X*.

If **NOFLSH** is set, the normal flush of the input and output queues associated with the **INTR**, **QUIT**, and **SUSP** characters will not be done.

If **TOSTOP** is set, the signal **SIGTTOU** is sent to a process that tries to write to its controlling terminal if it is not in the distinguished process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring **SIGTTOU** signals are excepted and allowed to produce output.

If **FLUSHO** is set, data written to the terminal will be discarded. This bit is set when the **FLUSH** character is typed. A program can cancel the effect of typing the **FLUSH** character by clearing **FLUSHO**.

If **PENDIN** is set, any input that has not yet been read will be reprinted when the next character arrives as input.

The initial line-discipline control value is **ISIG, ICANON, ECHO.**

Minimum and Timeout

The **MIN** and **TIME** values are described above under **Non-canonical Mode Input Processing**. The initial value of **MIN** is 1, and the initial value of **TIME** is 0.

Termio Structure

The System V **termio** structure is used by other **ioctl()** calls; it is defined by **<sys/termio.h>** as:

```
#define  NCC       8
struct   termio {
         unsigned  short  c_iflag;     /* input modes */
         unsigned  short  c_oflag;     /* output modes */
         unsigned  short  c_cflag;     /* control modes */
         unsigned  short  c_lflag;     /* local modes */
         char             c_line;      /* line discipline */
         unsigned  char   c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions for each function are as follows:

0	**VINTR**
1	**VQUIT**
2	**VERASE**
3	**VKILL**
4	**VEOF**
5	**VEOL**
6	**VEOL2**
7	reserved

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

Terminal Size

The number of lines and columns on the terminal's display (or page, in the case of printing terminals) is specified in the **winsize** structure, defined by **<sys/termios.h>**. Several **ioctl()** system calls that fetch or change these parameters use this structure:

```
struct winsize {
         unsigned short    ws_row;     /* rows, in characters */
         unsigned short    ws_col;     /* columns, in characters */
```

```
        unsigned short      ws_xpixel;   /* horizontal size, pixels - not used */
        unsigned short      ws_ypixel;   /* vertical size, pixels - not used */
};
```

Modem Lines

On special files representing serial ports, the modem control lines supported by the hardware can be read and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by **<sys/termios.h>**:

TIOCM_LE	0001	line enable
TIOCM_DTR	0002	data terminal ready
TIOCM_RTS	0004	request to send
TIOCM_ST	0010	secondary transmit
TIOCM_SR	0020	secondary receive
TIOCM_CTS	0040	clear to send
TIOCM_CAR	0100	carrier detect
TIOCM_RNG	0200	ring
TIOCM_DSR	0400	data set ready

TIOCM_CD is a synonym for **TIOCM_CAR**, and **TIOCM_RI** is a synonym for **TIOCM_RNG**.

Not all of these will necessarily be supported by any particular device; check the manual page for the device in question.

IOCTLS

The **ioctl**( ) calls supported by devices and STREAMS modules providing the **termios** interface are listed below. Some calls may not be supported by all devices or modules.

Unless otherwise noted for a specific **ioctl**( ) call, these functions are restricted from use by background processes. Attempts to perform these calls will cause the process group of the process performing the call to be sent a **SIGTTOU** signal. If the process is ignoring **SIGTTOU**, has **SIGTTOU** blocked, or is in the middle of process creation using **vfork**( ), the process will be allowed to perform the call and the **SIGTTOU** signal will not be sent.

TCGETS The argument is a pointer to a **termios** structure. The current terminal parameters are fetched and stored into that structure. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process.

TCSETS The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

TCSETSW The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

TCSETSF The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCGETA The argument is a pointer to a **termio** structure. The current terminal parameters are fetched, and those parameters that can be stored in a **termio** structure are stored into that structure. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process.

TCSETA The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change is immediate.

TCSETAW
: The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

TCSETAF
: The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCSBRK
: The argument is an **int** value. Wait for the output to drain. If the argument is 0, then send a break (zero-valued bits for 0.25 seconds). This define is available by **#include <sys/termio.h>**

TCXONC
: Start/stop control. The argument is an **int** value. If the argument is **TCOOFF** (0), suspend output; if **TCOON** (1), restart suspended output; if **TCIOFF** (2), suspend input; if **TCION** (3), restart suspended input.

TCFLSH
: The argument is an **int** value. If the argument is **TCIFLUSH** (0), flush the input queue; if **TCOFLUSH** (1), flush the output queue; if **TCIOFLUSH** (2), flush both the input and output queues.

TIOCEXCL
: The argument is ignored. Exclusive-use mode is turned on; no further opens are permitted until the file has been closed, or a **TIOCNXCL** is issued. The default on open of a terminal file is that exclusive use mode is off. This **ioctl()** is only available by **#include <sys/ttold.h>** .

TIOCNXCL
: The argument is ignored. Exclusive-use mode is turned off. This **ioctl()** is only available by **#include <sys/ttold.h>**.

TIOCSCTTY
: The argument is an **int**. The system will attempt to assign the terminal as the caller's controlling terminal (see **The Controlling Terminal** above). If the caller is not the super-user and/or the argument is not 1, all of the normal permission checks apply. If the caller is the super-user and the argument is 1 the terminal will be assigned as the controlling terminal even if the terminal was currently in use as a controlling terminal by another session. **getty**(8) uses this method to acquire controlling terminals for **login**(1) because there exists a possibility that a daemon process may obtain the console before **getty**(8).

TIOCGPGRP
: The argument is a pointer to an **int**. Set the value of that **int** to the process group ID of the distinguished process group associated with the terminal. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process. This **ioctl()** exists only for backward compatibility, use **tcgetpgrp**(3V).

TIOCSPGRP
: The argument is a pointer to an **int**. Associate the process group whose process group ID is specified by the value of that **int** with the terminal. The new process group value must be in the range of valid process group ID values, or it must be zero ("no process group"). Otherwise, the error **EINVAL** is returned. If any processes exist with a process ID or process group ID that is the same as the new process group value, then those processes must have the same real or saved user ID as the real or effective user ID of the calling process or be descendants of the calling process, or the effective user ID of the current process must be super-user. Otherwise, the error **EPERM** is returned. This **ioctl()** exists only for backward compatibility, use **tcsetpgrp()**, see **tcgetpgrp**(3V).

TIOCOUTQ
: The argument is a pointer to an **int**. Set the value of that **int** to the number of characters in the output stream that have not yet been sent to the terminal. This call is allowed from a background process.

TIOCSTI The argument is a pointer to a **char**. Pretend that character had been received as input.

TIOCGWINSZ The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is stored into that structure. This call is allowed from a background process.

TIOCSWINSZ The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a **SIGWINCH** signal is sent to the process group of the terminal.

TIOCMGET The argument is a pointer to an **int**. The current state of the modem status lines is fetched and stored in the **int** pointed to by the argument. This call is allowed from a background process.

TIOCMBIS The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.

TIOCMBIC The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.

TIOCMSET The argument is a pointer to an **int** containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

TIOCGSOFTCAR The argument is a pointer to an **int** whose value is 1 or 0, depending on whether the software carrier detect is turned on or off.

TIOCSSOFTCAR The argument is a pointer to an **int** whose value is 1 or 0. The value of the integer should be 0 to turn off software carrier, or 1 to turn it on.

SEE ALSO

csh(1), **login**(1), **stty**(1V), **fork**(2V), **getpgrp**(2V), **ioctl**(2), **open**(2V), **read**(2V), **sigvec**(2), **vfork**(2), **tcgetpgrp**(3V), **tty**(4), **ttytab**(5), **getty**(8), **init**(8), **ttysoftcar**(8)

NAME
tfs, TFS – translucent file service

CONFIG
options **TFS**

SYNOPSIS
#include <sys/mount.h>
mount("tfs", dir, M_NEWTYPE|flags, nfsargs);

DESCRIPTION
The translucent file service (TFS) supplies a copy-on-write filesystem allowing users to share file hierarchies while providing each user with a private hierarchy into which files are copied as they are modified. Consequently, users are isolated from each other's changes.

nfsargs specifies NFS style **mount**(2V) arguments, including the address of the file server (the **tfsd**(8)) and the file handle to be mounted. *dir* is the directory on which the TFS filesystem is to be mounted.

TFS allows a user to mount a private, writable filesystem in front of any number of public, read-only filesystems in such a way that the contents of the public filesystems remain visible behind the contents of the private filesystem. Any change made to a file that is being shared from a public filesystem will cause that file to be copied into the private filesystem, where the modification will be performed.

A directory in a TFS filesystem consists of a number of stacked directories. The searchpath TFS uses to look up a file in a directory corresponds to the stacking order: the TFS will search the "frontmost" directory first, then the directory behind it, and so on until the first occurrence of the file is found. Modifications to a file can be made only in the frontmost directory. TFS copies a file to the frontmost directory when the file is opened for writing with **open**(2V) or when its **stat**(2V) attributes are changed.

If a user removes a file which is not in the frontmost directory, TFS creates a *whiteout* entry in the frontmost directory and leaves the file intact in the back directory. This whiteout entry makes it appear that the file no longer exists, although the file can be reinstated in the directory by using the **unwhiteout**(1) command to remove the whiteout entry. The **lsw**(1) command lists whiteout entries.

TFS filesystems are served by the **tfsd**(8). A TFS filesystem is mounted on a directory by making a **TFS_MOUNT** protocol request of the **tfsd**, specifying the directories that are to be stacked. The **tfsd** responds with a file handle, which the client then supplies to the **mount**(2V) system call, along with the address of the **tfsd**.

SEE ALSO
lsw(1), **unwhiteout**(1), **mount**(2V), **tfsd**(8), **mount_tfs**(8)

NAME

timod – Transport Interface cooperating STREAMS module

CONFIG

pseudo-device tim64

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library (see Section 3). The **timod** module converts a set of **ioctl**(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The **timod** module must be pushed onto only a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the **ioctl()** commands described below, are transparently passed to the neighboring STREAMS module or driver. The messages generated from the following **ioctl()** commands are recognized and processed by the **timod** module. The format of the **ioctl()** call is:

Where, on issuance, **size** is the size of the appropriate TI message to be sent to the transport provider and on return **size** is the size of the appropriate TI message from the transport provider in response to the issued TI message. **buf** is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in **<sys/tihdr.h>**. The possible values for the **cmd** field are:

TI_BIND
: Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND **ioctl()** is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the **ioctl()** is equivalent to the TI message type T_BIND_ACK.

TI_UNBIND
: Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND **ioctl()** is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the **ioctl()** is equivalent to the TI message type T_OK_ACK.

TI_GETINFO
: Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO **ioctl()** is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the **ioctl()** is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT
: Get, set or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT **ioctl()** is equivalent to the TI message type T_OPTMGMT_REQ and the message returned by the successful completion of the **ioctl()** is equivalent to the TI message type T_OPTMGMT_ACK.

SEE ALSO

tirdwr(4)

Network Programming

DIAGNOSTICS

If the **ioctl()** system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in **<sys/tiuser.h>**. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in **<sys/errno.h>** (see **intro**(2)).

NAME

tirdwr – Transport Interface read/write interface STREAMS module

CONFIG

pseudo-device tirw64

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3). This alternate interface allows a user to communicate with the transport protocol provider using the **read**(2V) and **write**(2V) system calls. The **putmsg**(2) and **getmsg**(2) system calls may also be used. However, **putmsg**( ) and **getmsg**( ) can only transfer data messages between user and *stream*.

The **tirdwr** module must only be pushed (see I_PUSH in **streamio**(4)) onto a **stream** terminated by a transport protocol provider which supports the TI. After the **tirdwr** module has been pushed onto a *stream*, none of the Transport Interface functions can be used. Subsequent calls to TI functions cause an error on the *stream*. Once the error is detected, subsequent system calls on the **stream** return an error with **errno** set to EPROTO.

The following are the actions taken by the **tirdwr** module when pushed on the **stream**, popped (see I_POP in **streamio**(4)) off the *stream*, or when data passes through it.

push When the module is pushed onto a **stream**, it checks any existing data destined for the user to ensure that only regular data messages are present. It ignores any messages on the **stream** that relate to process management, such as messages that generate signals to the user processes associated with the *stream*. If any other messages are present, the I_PUSH returns an error with **errno** set to EPROTO.

write The module takes the following actions on data that originated from a **write**( ) system call:

All messages with the exception of messages that contain control portions (see **putmsg**(2) and **getmsg**(2)) are transparently passed onto the module's downstream neighbor.

Any zero length data message is freed by the module and is not passed onto the module's downstream neighbor.

Any message with a control portion generates an error, and any further system calls associated with the **stream** fail with **errno** set to EPROTO.

read The module takes the following actions on data that originated from the transport protocol provider:

All messages with the exception of those that contain control portions (see the **putmsg** and **getmsg** system calls) are transparently passed onto the module's upstream neighbor.

The action taken on messages with control portions is as follows:

- Messages that represent expedited data generate an error. All further system calls associated with the **stream** fail with **errno** set to EPROTO.
- Any data messages with control portions have the control portions removed from the message prior to passing the message on to the upstream neighbor.
- Messages that represent an orderly release indication from the transport provider generate a zero length data message, indicating the end of file, which are sent to the reader of the *stream*. The orderly release message itself is freed by the module.
- Messages that represent an abortive disconnect indication from the transport provider cause all further **write()** and **putmsg()** calls to fail with **errno** set to ENXIO. All further **read**( ) and **getmsg**( ) calls return zero length data (indicating an EOF) once all previous data has been read.

- With the exception of the above rules, all other messages with control portions generate an error and all further system calls associated with the **stream** fail with **errno** set to EPROTO.

Any zero length data messages are freed by the module and they are not passed onto the module's upstream neighbor.

pop When the module is popped off the **stream** or the **stream** is closed, the module takes the following action:

If an orderly release indication has been previously received, then an orderly release request is sent to the remote side of the transport connection.

SEE ALSO

intro(2), **getmsg**(2), **putmsg**(2), **read**(2V), **write**(2V), **intro**(3), **streamio**(4), **timod**(4)

Network Programming

NAME

tm – Tapemaster 1/2 inch tape controller

CONFIG — SUN-3, SUN-3x SYSTEMS

controller tm0 at vme16d16 ? csr 0xa0 priority 3 vector tmintr 0x60
controller tm1 at vme16d16 ? csr 0xa2 priority 3 vector tmintr 0x61
tape mt0 at tm0 drive 0 flags 1
tape mt0 at tm1 drive 0 flags 1

DESCRIPTION

The Tapemaster tape controller controls Pertec-interface 1/2" tape drives such as the CDC Keystone, providing a standard tape interface to the device, see **mtio**(4). This controller supports single-density or speed drives.

The **tm** driver supports the character device interface. The driver returns an ENOTTY error on unsupported ioctls.

The **tm** driver does not support the backspace file to beginning of file (MTNBSF n) command. The equivalent positioning can be obtained by using MTBSF (n+1) followed by MTFSF 1.

Half-inch reel tape devices do not support the retension ioctl.

FILES

/dev/rmt*	rewinding
/dev/nrmt*	non-rewinding

SEE ALSO

mt(1), **tar**(1), **mtio**(4), **st**(4S), **xt**(4S)

BUGS

The Tapemaster controller does not provide for byte-swapping and the resultant system overhead prevents streaming transports from streaming.

The system should remember which controlling terminal has the tape drive open and write error messages to that terminal rather than on the console.

The Tapemaster controller is not supported on Sun-4 systems.

WARNINGS

The Tapemaster interface will not be supported in a future release. The Xylogics 472 controller and **xt** driver replace the Tapemaster controller and **tm** driver.

NAME

tmpfs – memory based filesystem

CONFIG

options TMPFS

SYNOPSIS

#include <sys/mount.h>
mount ("tmpfs", dir, M_NEWTYPE | flags, args);

DESCRIPTION

tmpfs is a memory based filesystem which uses kernel resources relating to the VM system and page cache as a filesystem. Once mounted, a **tmpfs** filesystem provides standard file operations and semantics. **tmpfs** is so named because files and directories are not preserved across reboot or unmounts, all files residing on a **tmpfs** filesystem that is unmounted will be lost.

tmpfs filesystems are mounted either with the command:

mount –t tmp swap *directory-name*

or by placing the line

swap *directory-name* **tmp rw 0 0**

in your **/etc/fstab** file and using the **mount**(8) command as normal. The **/etc/rc.local** file contains commands to mount a **tmpfs** filesystem on **/tmp** at multi-user startup time but is by default commented out. To mount a **tmpfs** filesystem on **/tmp** (maximizing possible performance improvements), add the above line to **/etc/fstab** and uncomment the following line in **/etc/rc.local**:

#mount /tmp

tmpfs is designed as a performance enhancement which is achieved by cacheing the writes to files residing on a **tmpfs** filesystem. Performance improvements are most noticeable when a large number of short lived files are written and accessed on a **tmpfs** filesystem. Large compilations with **tmpfs** mounted on **/tmp** are a good example of this.

Users of **tmpfs** should be aware of some tradeoffs involved in mounting a **tmpfs** filesystem. The resources used by **tmpfs** are the same as those used when commands are executed (for example, swap space allocation). This means that a large sized or number of **tmpfs** files can affect the amount of space left over for programs to execute. Likewise, programs requiring large amounts of memory use up the space available to **tmpfs**. Users running into these constraints (for example, running out of space on **tmpfs**) can allocate more swap space by using the **swapon**(8) command.

Normal filesystem writes are scheduled to be written to a permanent storage medium along with all control information associated with the file (for example, modification time, file permissions). **tmpfs** control information resides only in memory and never needs to be written to permanent storage. File data remains in core until memory demands are sufficient to cause pages associated with **tmpfs** to be reused at which time they are copied out to swap.

SEE ALSO

df(1V), **mount**(2V), **umount**(2V), **fstab**(5), **mount**(8), **swapon**(8)

System Services Overview,
System and Network Administration

NOTES

swapon to a **tmpfs** file is not supported.

df(1V) output is of limited accuracy since a **tmpfs** filesystem size is not static and the space available to **tmpfs** is dependent on the swap space demands of the entire system.

DIAGNOSTICS

If **tmpfs** runs out of space, one of the following messages will be printed to the console.

directory: **file system full, anon reservation exceeded**

directory: **file system full, anon allocation exceeded**

A page could not be allocated while writing to a file. This can occur if **tmpfs** is attempting to write more than it is allowed, or if currently executing programs are using a lot of memory. To make more space available, remove unneccessary files, exit from some programs, or allocate more swap space using **swapon**(8).

directory: **file system full, kmem_alloc failure**

tmpfs ran out of physical memory while attempting to create a new file or directory. Remove unneccesary files or directories or install more physical memory.

WARNINGS

A **tmpfs** filesystem should *not* be mounted on **/var/tmp**, this directory is used by **vi**(1) for preserved files.

Files and directories on a **tmpfs** filesystem are not preserved across reboots or unmounts. Command scripts or programs which count on this will not work as expected.

NAME

ttcompat – V7 and 4BSD STREAMS compatibility module

CONFIG

None; included by default.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>

ioctl(fd, I_PUSH, "ttcompat");
```

DESCRIPTION

ttcompat is a STREAMS module that translates the **ioctl** calls supported by the older Version 7 and 4BSD terminal drivers into the **ioctl** calls supported by the **termio**(4) interface. All other messages pass through this module unchanged; the behavior of **read** and **write** calls is unchanged, as is the behavior of **ioctl** calls other than the ones supported by **ttcompat**.

Normally, this module is automatically pushed onto a stream when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the **termio** interface be supported by the modules and driver downstream. The **TCGETS**, **TCSETS**, and **TCSETSF** **ioctl** calls must be supported; if any information set or fetched by those **ioctl** calls is not supported by the modules and driver downstream, some of the V7/4BSD functions may not be supported. For example, if the **CBAUD** bits in the **c_cflag** field are not supported, the functions provided by the **sg_ispeed** and **sg_ospeed** fields of the **sgttyb** structure (see below) will not be supported. If the **TCFLSH ioctl** is not supported, the function provided by the **TIOCFLUSH ioctl** will not be supported. If the **TCXONC ioctl** is not supported, the functions provided by the **TIOCSTOP** and **TIOCSTART ioctl** calls will not be supported. If the **TIOCMBIS** and **TIOCMBIC ioctl** calls are not supported, the functions provided by the **TIOCSDTR** and **TIOCCDTR ioctl** calls will not be supported.

The basic **ioctl** calls use the **sgttyb** structure defined by <sys/ioctl.h>:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        short   sg_flags;
};
```

The **sg_ispeed** and **sg_ospeed** fields describe the input and output speeds of the device, and reflect the values in the **c_cflag** field of the **termio** structure. The **sg_erase** and **sg_kill** fields of the argument structure specify the erase and kill characters respectively, and reflect the values in the **VERASE** and **VKILL** members of the **c_cc** field of the **termio** structure.

The **sg_flags** field of the argument structure contains several flags that determine the system's treatment of the terminal. They are mapped into flags in fields of the terminal state, represented by the **termio** structure.

Delay type 0 is always mapped into the equivalent delay type 0 in the **c_oflag** field of the **termio** structure. Other delay mappings are performed as follows:

sg_flags	c_oflag
BS1	BS1
FF1	VT1
CR1	CR2
CR2	CR3
CR3	not supported
TAB1	TAB1
TAB2	TAB2

XTABS	**TAB3**
NL1	**ONLRET|CR1**
NL2	**NL1**

If previous **TIOCLSET** or **TIOCLBIS ioctl** calls have not selected **LITOUT** or **PASS8** mode, and if **RAW** mode is not selected, the **ISTRIP** flag is set in the **c_iflag** field of the **termio** structure, and the **EVENP** and **ODDP** flags control the parity of characters sent to the terminal and accepted from the terminal:

0 Parity is not to be generated on output or checked on input; the character size is set to **CS8** and the **PARENB** flag is cleared in the **c_cflag** field of the **termio** structure.

EVENP Even parity characters are to be generated on output and accepted on input; the **INPCK** flag is set in the **c_iflag** field of the **termio** structure, the character size is set to **CS7** and the **PARENB** flag is set in the **c_cflag** field of the **termio** structure.

ODDP Odd parity characters are to be generated on output and accepted on input; the **INPCK** flag is set in the **c_iflag** field, the character size is set to **CS7** and the **PARENB** and **PARODD** flags are set in the **c_cflag** field of the **termio** structure.

EVENP | ODDP
Even parity characters are to be generated on output and characters of either parity are to be accepted on input; the **INPCK** flag is cleared in the **c_iflag** field, the character size is set to **CS7** and the **PARENB** flag is set in the **c_cflag** field of the **termio** structure.

The **RAW** flag disables all output processing (the **OPOST** flag in the **c_oflag** field, and the **XCASE** flag in the **c_lflag** field, are cleared in the **termio** structure) and input processing (all flags in the **c_iflag** field other than the **IXOFF** and **IXANY** flags are cleared in the **termio** structure). 8 bits of data, with no parity bit, are accepted on input and generated on output; the character size is set to **CS8** and the **PARENB** and **PARODD** flags are cleared in the **c_cflag** field of the **termio** structure. The signal-generating and line-editing control characters are disabled by clearing the **ISIG** and **ICANON** flags in the **c_lflag** field of the **termio** structure.

The **CRMOD** flag turn input RETURN characters into NEWLINE characters, and output and echoed NEWLINE characters to be output as a RETURN followed by a LINEFEED. The **ICRNL** flag in the **c_iflag** field, and the **OPOST** and **ONLCR** flags in the **c_oflag** field, are set in the **termio** structure.

The **LCASE** flag maps upper-case letters in the ASCII character set to their lower-case equivalents on input (the **IUCLC** flag is set in the **c_iflag** field), and maps lower-case letters in the ASCII character set to their upper-case equivalents on output (the **OLCUC** flag is set in the **c_oflag** field). Escape sequences are accepted on input, and generated on output, to handle certain ASCII characters not supported by older terminals (the **XCASE** flag is set in the **c_lflag** field).

Other flags are directly mapped to flags in the **termio** structure:

sg_flags	flags in **termio** structure
CBREAK	complement of **ICANON** in **c_lflag** field
ECHO	**ECHO** in **c_lflag** field
TANDEM	**IXOFF** in **c_iflag** field

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by **<sys/ioctl.h>**:

```
struct tchars {
        char    t_intrc;        /* interrupt */
        char    t_quitc;        /* quit */
        char    t_startc;       /* start output */
        char    t_stopc;        /* stop output */
        char    t_eofc;         /* end-of-file */
        char    t_brkc;         /* input delimiter (like nl) */
};
```

The characters are mapped to members of the **c_cc** field of the **termio** structure as follows:

tchars	**c_cc** index
t_intrc	**VINTR**
t_quitc	**VQUIT**
t_startc	**VSTART**
t_stopc	**VSTOP**
t_eofc	**VEOF**
t_brkc	**VEOL**

Also associated with each terminal is a local flag word, specifying flags supported by the new 4BSD terminal interface. Most of these flags are directly mapped to flags in the **termio** structure:

local flags	flags in **termio** structure
LCRTBS	not supported
LPRTERA	**ECHOPRT** in the **c_lflag** field
LCRTERA	**ECHOE** in the **c_lflag** field
LTILDE	not supported
LTOSTOP	**TOSTOP** in the **c_lflag** field
LFLUSHO	**FLUSHO** in the **c_lflag** field
LNOHANG	**CLOCAL** in the **c_cflag** field
LCRTKIL	**ECHOKE** in the **c_lflag** field
LCTLECH	**CTLECH** in the **c_lflag** field
LPENDIN	**PENDIN** in the **c_lflag** field
LDECCTQ	complement of **IXANY** in the **c_iflag** field
LNOFLSH	**NOFLSH** in the **c_lflag** field

Another structure associated with each terminal is the **ltchars** structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```
struct ltchars {
        char    t_suspc;        /* stop process signal */
        char    t_dsuspc;       /* delayed stop process signal */
        char    t_rprntc;       /* reprint line */
        char    t_flushc;       /* flush output (toggles) */
        char    t_werasc;       /* word erase */
        char    t_lnextc;       /* literal next character */
};
```

The characters are mapped to members of the **c_cc** field of the **termio** structure as follows:

ltchars	**c_cc** index
t_suspc	**VSUSP**
t_dsuspc	**VDSUSP**
t_rprntc	**VREPRINT**
t_flushc	**VDISCARD**
t_werasc	**VWERASE**
t_lnextc	**VLNEXT**

IOCTLS

ttcompat responds to the following **ioctl** calls. All others are passed to the module below.

TIOCGETP The argument is a pointer to an **sgttyb** structure. The current terminal state is fetched; the appropriate characters in the terminal state are stored in that structure, as are the input and output speeds. The values of the flags in the **sg_flags** field are derived from the flags in the terminal state and stored in the structure.

TIOCSETP The argument is a pointer to an **sgttyb** structure. The appropriate characters and input and output speeds in the terminal state are set from the values in that structure, and the flags in the terminal state are set to match the values of the flags in the **sg_flags** field of that structure. The state is changed with a **TCSETSF** *ioctl*, so that the interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN The argument is a pointer to an **sgttyb** structure. The terminal state is changed as **TIOCSETP** would change it, but a **TCSETS ioctl** is used, so that the interface neither delays nor discards input.

TIOCHPCL The argument is ignored. The **HUPCL** flag is set in the **c_cflag** word of the terminal state.

TIOCFLUSH The argument is a pointer to an **int** variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the **int** is treated as the logical OR of the **FREAD** and **FWRITE** flags defined by **<sys/file.h>**; if the **FREAD** bit is set, all characters waiting in input queues are flushed, and if the **FWRITE** bit is set, all characters waiting in output queues are flushed.

TIOCSBRK The argument is ignored. The break bit is set for the device.

TIOCCBRK The argument is ignored. The break bit is cleared for the device.

TIOCSDTR The argument is ignored. The Data Terminal Ready bit is set for the device.

TIOCCDTR The argument is ignored. The Data Terminal Ready bit is cleared for the device.

TIOCSTOP The argument is ignored. Output is stopped as if the **STOP** character had been typed.

TIOCSTART The argument is ignored. Output is restarted as if the **START** character had been typed.

TIOCGETC The argument is a pointer to an **tchars** structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.

TIOCSETC The argument is a pointer to an **tchars** structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

TIOCLGET The argument is a pointer to an **int**. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the **int** pointed to by the argument.

TIOCLBIS The argument is a pointer to an **int** whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLBIC The argument is a pointer to an **int** whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLSET The argument is a pointer to an **int** containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word.

TIOCGLTC The argument is a pointer to an **ltchars** structure. The values of the appropriate characters in the terminal state are stored in that structure.

TIOCSLTC The argument is a pointer to an **ltchars** structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

SEE ALSO

ioctl(2), **termio**(4)

NAME

tty – controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the controlling terminal of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

IOCTLS

In addition to the **ioctl( )** requests supported by the device that **tty** refers to, the following **ioctl( )** request is supported:

TIOCNOTTY Detach the current process from its controlling terminal, and remove it from its current process group, without attaching it to a new process group (that is, set its process group ID to zero). This **ioctl( )** call only works on file descriptors connected to **/dev/tty**; this is used by daemon processes when they are invoked by a user at a terminal. The process attempts to open **/dev/tty**; if the open succeeds, it detaches itself from the terminal by using **TIOCNOTTY**, while if the open fails, it is obviously not attached to a terminal and does not need to detach itself.

FILES

/dev/tty

SEE ALSO

termio(4)

NAME

udp – Internet User Datagram Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the **SOCK_DGRAM** abstraction for the Internet protocol family. It is layered directly above the Internet Protocol (IP). UDP sockets are connectionless, and are normally used with the **sendto**, **sendmsg**, **recvfrom**, and **recvmsg** system calls (see **send**(2) and **recv**(2)). If the **connect**(2) system call is used to fix the destination for future packets, then the **recv**(2) or **read**(2V) and **send**(2) or **write**(2V) system calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (TCP). Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication. Note: the UDP port number space is separate from the TCP port number space (that is, a UDP port may not be "connected" to a TCP port). The **bind**(2) system call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the **bind** call by using the special value **INADDR_ANY**. If the **bind** call is not done, a local IP address and port number will be assigned to each packet as it is sent. Broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent. Broadcasts may only be sent by the super-user.

Options at the IP level may be used with UDP; see **ip**(4P).

There are a variety of ways that a UDP packet can be lost or discarded, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

UDP processes Internet Control Message Protocol (ICMP) error messages received in response to UDP packets it has sent. See **icmp**(4P). ICMP "source quench" messages are ignored. ICMP "destination unreachable," "time exceeded" and "parameter problem" messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

ERRORS

A socket operation may fail if:

EISCONN — A **connect** operation was attempted on a socket on which a **connect** operation had already been performed, and the socket could not be successfully disconnected before making the new connection.

EISCONN — A **sendto** or **sendmsg** operation specifying an address to which the message should be sent was attempted on a socket on which a **connect** operation had already been performed.

ENOTCONN — A **send** or **write** operation, or a **sendto** or **sendmsg** operation not specifying an address to which the message should be sent, was attempted on a socket on which a **connect** operation had not already been performed.

EADDRINUSE — A **bind** operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EADDRNOTAVAIL — A **bind** operation was attempted on a socket with a network address for which no network interface exists.

EINVAL	A **sendmsg** operation with a non-NULL **msg_accrights** was attempted.
EACCES	A **bind** operation was attempted with a "reserved" port number and the effective user ID of the process was not super-user.
ENOBUFS	The system ran out of memory for internal data structures.

SEE ALSO

bind(2), **connect**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **icmp**(4P), **inet**(4F), **ip**(4P), **tcp**(4P)

Postel, Jon, *User Datagram Protocol*, RFC 768, Network Information Center, SRI International, Menlo Park, Calif., August 1980. (Sun 800-1054-01)

BUGS

SIOCSHIWAT and **SIOCGHIWAT** ioctl's to set and get the high water mark for the socket queue, and so that it can be changed from 2048 bytes to be larger or smaller, have been defined (in **sys/ioctl.h**) but not implemented.

Something sensible should be done with ICMP source quench error messages if the socket is bound to a peer socket.

NAME

unix – UNIX domain protocol family

DESCRIPTION

The Unix Domain protocol family provides support for socket-based communication between processes running on the local host. While both **SOCK_STREAM** and **SOCK_DGRAM** types are supported, the **SOCK_STREAM** type often provides faster performance. Pipes, for instance, are built on Unix Domain **SOCK_STREAM** sockets.

Unix Domain **SOCK_DGRAM** sockets (also called datagram sockets) exist primarily for reasons of orthogonality under the BSD socket model. However, the overhead of reading or writing data is higher for the (connectionless) datagram sockets.

Unix Domain addresses are pathnames. In other words, two independent processes can communicate by specifying the same pathname as their communications rendezvous point. The **bind**(2) operation creates a special entry in the file system of type socket. If that pathname already exists (as a socket from a previous **bind**( ) operation, or as some other file system type), **bind**( ) will fail.

Sockets in the Unix domain protocol family use the following addressing structure:

```
struct sockaddr_un {
        short    sun_family;
        u_short sun_path[108];
};
```

To create or reference a Unix Domain socket, the **sun_family** field should be set to **AF_UNIX** and the **sun_path** array should contain the path name of a rendezvous point.

Although Unix Domain sockets are faster than Internet Domain sockets for communication between local processes, the advantage of the additional flexibility afforded by the latter may outweigh performance issues. Where inter-process communication thoughput is critical, a shared memory approach may be preferred.

Since there are no protocol families associated with Unix Domain sockets, the protocol argument to **socket**(2) should be zero.

When setting up a Unix Domain socket, the *length* argument to the **bind**( ) call is the amount of space within the **sockaddr_un** structure, not including the pathname delimiter. One way to specify the length is:

> **sizeof**(*addr.sun_family*) + **strlen**(*path*) where *addr* is a structure of type **sockaddr_un**, and *path* is a pointer to the pathname.

The limit of 108 characters is an artifact of the implementation.

Since closing a Unix Domain socket does not make the file system entry go away, an application should remove the entry using **unlink**(2V), when finished.

SEE ALSO

bind(2), **socket**(2), **unlink**(2V)

Network Programming

NAME
vd – loadable modules interface

CONFIG
None; included with **options VDDRV**

DESCRIPTION
This pseudo-device provides kernel support for loadable modules. It is used exclusively by the **modload**(8), **modunload**(8), and **modstat**(8) utilities. Other programs should not use it.

FILES
/dev/vd

SEE ALSO
modload(), modunload(), modstat()

WARNINGS
The interface provided by **vd** is subject to change without notice.

NAME

vpc – Systech VPC-2200 Versatec printer/plotter and Centronics printer interface

CONFIG

device vpc0 at vme16d16 ? csr 0x480 priority 2 vector vpcintr 0x80
device vpc1 at vme16d16 ? csr 0x500 priority 2 vector vpcintr 0x81

AVAILABILITY

Sun-3, Sun-3/80 and Sun-4 systems only.

DESCRIPTION

This Sun interface to the Versatec printer/plotter and to Centronics printers is supported by the Systech parallel interface board, an output-only byte-wide DMA device. The device has one channel for Versatec devices and one channel for Centronics devices, with an optional long lines interface for Versatec devices.

Devices attached to this interface are normally handled by the line printer spooling system and should not be accessed by the user directly.

Opening the device **/dev/vpc0** or **/dev/lp0** may yield one of two errors: ENXIO indicates that the device is already in use; EIO indicates that the device is offline.

The Versatec printer/plotter operates in either print or plot mode. To set the printer into plot mode you should include **<sys/vcmd.h>** and use the **ioctl**(2) call:

```
ioctl(f, VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[ ] = { VPLOT, 0, 0 };
```

When going back into print mode from plot mode you normally eject paper by sending it an EOT after putting into print mode:

```
int prtmd[ ] = { VPRINT, 0, 0 };
...
fflush (vpc);
f = fileno(vpc);
ioctl(f, VSETSTATE, prtmd);
write(f, "\04", 1);
```

FILES

/dev/vpc0
/dev/lp0

SEE ALSO

ioctl(2), **setbuf**(3V)

BUGS

If you use the standard I/O library on the Versatec, be sure to explicitly set a buffer using **setbuf**(3V), since the library will not use buffered output by default, and will run very slowly.

NAME

win – Sun window system

CONFIG

pseudo-device win*number*
pseudo-device dtop*number*

DESCRIPTION

The **win** pseudo-device accesses the system drivers supporting the Sun window system. *number*, in the device description line above, indicates the maximum number of windows supported by the system. *number* is set to 128 in the GENERIC system configuration file used to generate the kernel used in Sun systems as they are shipped. The *dtop* pseudo-device line indicates the number of separate ''desktops'' (frame buffers) that can be actively running the Sun window system at once. In the GENERIC file, this number is set to 4.

Each window in the system is represented by a **/dev/win*** device. The windows are organized as a tree with windows being subwindows of their parents, and covering/covered by their siblings. Each window has a position in the tree, a position on a display screen, an input queue, and information telling what parts of it are exposed.

The window driver multiplexes keyboard and mouse input among the several windows, tracks the mouse with a cursor on the screen, provides each window access to information about what parts of it are exposed, and notifies the manager process for a window when the exposed area of the window changes so that the window may repair its display.

Full information on the window system functions is given in the *SunView System Programmer's Guide.*

FILES

/dev/win[0-9]
/dev/win[0-9][0-9]

SEE ALSO

SunView System Programmer's Guide

NAME

xd – Disk driver for Xylogics 7053 SMD Disk Controller

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

controller xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk xd0 at xdc0 drive 0
disk xd1 at xdc0 drive 1
disk xd2 at xdc0 drive 2
disk xd3 at xdc0 drive 3
disk xd4 at xdc1 drive 0
disk xd5 at xdc1 drive 1
disk xd6 at xdc1 drive 2
disk xd7 at xdc1 drive 3
disk xd8 at xdc2 drive 0
disk xd9 at xdc2 drive 1
disk xd10 at xdc2 drive 2
disk xd11 at xdc2 drive 3
disk xd12 at xdc3 drive 0
disk xd13 at xdc3 drive 1
disk xd14 at xdc3 drive 2
disk xd15 at xdc3 drive 3

The four **controller** lines given in the synopsis section above specify the first, second, third, and fourth Xylogics 7053 SMD disk controller in a Sun system.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, and so on. The standard device names begin with **xd** followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra **r**.

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise **directory**(3V) calls should specify a multiple of 512 bytes.

If **flags 0x1** is specified, the overlapped seeks feature for that drive is turned off. Note: to be effective, the flag must be set on all drives for a specific controller. This action is necessary for controllers with older firmware, which have bugs preventing overlapped seeks from working properly.

DISK SUPPORT

This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

The **xd?a** partition is normally used for the root file system on a disk, the xd?b partition as a paging area, and the **xd?c** partition for pack-pack copying (it normally maps the entire disk). The rest of the disk is normally the **xd?g** partition.

FILES

/dev/xd[0-7][a-h]	block files
/dev/rxd[0-7][a-h]	raw files

SEE ALSO

lseek(2V), **read**(2V), **write**(2V), **directory**(3V), **dkio**(4S)

DIAGNOSTICS

xdc*n*: self test error
Self test error in controller, see the Maintenance and Reference Manual.

xd*n*: unable to read bad sector
The bad sector forwarding information for the disk could not be read.

xd*n*: initialization failed
The drive could not be successfully initialized.

xd*n*: unable to read label
The drive geometry/partition table information could not be read.

xd*n*: Corrupt label
The geometry/partition label checksum was incorrect.

xd*n*: offline
A drive ready status is no longer detected, so the unit has been logically removed from the system. If the drive ready status is restored, the unit will automatically come back online the next time it is accessed.

xd*nc*: *cmd how* **(***msg***) blk** *#n abs blk #n*
A command such as read or write encountered an error condition (*how*): either it *failed*, the controller was *reset*, the unit was *restored*, or an operation was *retry*'ed. The *msg* is derived from the error number given by the controller, indicating a condition such as "drive not ready(rq, "sector not found" or "disk write protected". The *blk #* is the sector in error relative to the beginning of the partition involved. The *abs blk #* is the absolute block number of the sector in error. Some fields of the error message may be missing since the information is not always available.

BUGS

In raw I/O **read**(2V) and **write**(2V) truncate file offsets to 512-byte block boundaries, and **write**(2V) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read**(2V), **write**(2V) and **lseek**(2V) should always deal in 512-byte multiples.

Older revisions of the firmware do not properly support overlapped seeks. This will only affect systems with multiple disks on a single controller. If a large number of "zero sector count" errors appear, you should use the **flags** field to disable overlapped seeks.

NAME

xt – Xylogics 472 1/2 inch tape controller

CONFIG — SUN-3, SUN-4 SYSTEMS

controller xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape xt0 at xtc0 drive 0 flags 1
tape xt1 at xtc1 drive 0 flags 1

DESCRIPTION

The Xylogics 472 tape controller controls Pertec-interface 1/2" tape drives such as the Fujitsu M2444 and the CDC Keystone III, providing a standard tape interface to the device see **mtio**(4). This controller is used to support high speed or high density drives, which are not supported effectively by the older Tapemaster controller (see **tm**(4S)).

The flags field is used to control remote density select operation: a 0 specifies no remote density selection is to be attempted, a 1 specifies that the Pertec density-select line is used to toggle between high and low density; a 2 specifies that the Pertec speed-select line is used to toggle between high and low density. The default is 1, which is appropriate for the Fujitsu M2444, the CDC Keystone III (92185) and the Telex 9250. In no case will the controller select among more than 2 densities.

The **xt** driver supports the character device interface.

EOT Handling

The user will be notified of end of tape (EOT) on write by a 0 byte count returned the first time this is attempted. This write must be retried by the user. Subsequent writes will be successful until the tape winds off the reel. Read past EOT is transparent to the user.

Ioctls

Not all devices support all ioctls. The driver returns an ENOTTY error on unsupported ioctls.

1/2" tape devices do not support the tape retension function.

FILES

/dev/rmt0	low density operation, typically 1600 bpi
/dev/rmt8	high density operation, typically 6250 bpi
/dev/nrmt*	non-rewinding

SEE ALSO

mt(1), **tar**(1), **mtio**(4), **st**(4S), **suninstall**(8)

BUGS

Record sizes are restricted to an even number of bytes.

Absolute file positioning is not fully supported; it is only meant to be used by **suninstall**(8).

NAME

xy – Disk driver for Xylogics 450 and 451 SMD Disk Controllers

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

controller xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk xy0 at xyc0 drive 0
disk xy1 at xyc0 drive 1
disk xy2 at xyc1 drive 0
disk xy3 at xyc1 drive 1

The two **controller** lines given in the synopsis sections above specify the first and second Xylogics 450 or 451 SMD disk controller in a Sun system.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, and so on. The standard device names begin with **xy** followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character '**?**' stands here for a drive number in the range 0-7.

The block files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra **r**.

When using raw I/O, transfer counts should be multiples of 512 bytes (the size of a disk sector). Likewise, when using **lseek**(2V) to specify block offsets from which to perform raw I/O, the logical offset should also be a multiple of 512 bytes.

Due to word ordering differences between the disk controller and Sun computers, user buffers that are used for raw I/O must not begin on odd byte boundaries.

If **flags 0x1** is specified, the overlapped seeks feature for that drive is turned off. Note: to be effective, the flag must be set on all drives for a specific controller. This action is necessary for controllers with older firmware, which have bugs preventing overlapped seeks from working properly.

DISK SUPPORT

This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

The **xy?a** partition is normally used for the root file system on a disk, the **xy?b** partition as a paging area, and the **xy?c** partition for pack-pack copying (it normally maps the entire disk). The rest of the disk is normally the **xy?g** partition.

FILES

/dev/xy[0-7][a-h]	block files
/dev/rxy[0-7][a-h]	raw files

SEE ALSO

lseek(2V), **read**(2V), **directory**(3V), **write**(2V), **dkio**(4S)

DIAGNOSTICS

xyc*n* : self test error
Self test error in controller, see the Maintenance and Reference Manual.

xyc*n*: WARNING: *n* bit addresses
The controller is strapped incorrectly. Sun systems use 20-bit addresses for Multibus based systems and 24-bit addresses for VMEbus based systems.

xy*n* : unable to read bad sector info
The bad sector forwarding information for the disk could not be read.

xy*n* and xy*n* are of same type (*n*) with different geometries.
The 450 and 451 do not support mixing the drive types found on these units on a single controller.

xy*n* : initialization failed
The drive could not be successfully initialized.

xy*n* : unable to read label
The drive geometry/partition table information could not be read.

xy*n* : Corrupt label
The geometry/partition label checksum was incorrect.

xy*n* : offline
A drive ready status is no longer detected, so the unit has been logically removed from the system. If the drive ready status is restored, the unit will automatically come back online the next time it is accessed.

xy*nc*: *cmd how* (*msg*) **blk** *#n abs blk #n*
A command such as read or write encountered an error condition (*how*): either it *failed*, the controller was *reset*, the unit was *restored*, or an operation was *retry*'ed. The *msg* is derived from the error number given by the controller, indicating a condition such as "drive not ready", "sector not found" or "disk write protected". The *blk #* is the sector in error relative to the beginning of the partition involved. The *abs blk #* is the absolute block number of the sector in error. Some fields of the error message may be missing since the information is not always available.

BUGS

In raw I/O **read**(2V) and **write**(2V) truncate file offsets to 512-byte block boundaries, and **write**(2V) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read**(2V), **write**(2V) and **lseek**(2V) should always deal in 512-byte multiples.

Older revisions of the firmware do not properly support overlapped seeks. This will only affect systems with multiple disks on a single controller. If a large number of "zero sector count" errors appear, you should use the **flags** field to disable overlapped seeks.

NAME

zero – source of zeroes

SYNOPSIS

None; included with standard system.

DESCRIPTION

A zero special file is a source of zeroed unnamed memory.

Reads from a zero special file always return a buffer full of zeroes. The file is of infinite length.

Writes to a zero special file are always successful, but the data written is ignored.

Mapping a zero special file creates a zero-initialized unnamed memory object of a length equal to the length of the mapping and rounded up to the nearest page size as returned by **getpagesize**(2). Multiple processes can share such a zero special file object provided a common ancestor mapped the object **MAP_SHARED**.

FILES

/dev/zero

SEE ALSO

fork(2V), **getpagesize**(2), **mmap**(2)

NAME

zs – Zilog 8530 SCC serial communications driver

CONFIG — SUN-3 SYSTEM

device zs0 at obio ? csr 0x20000 flags 3 priority 3
device zs1 at obio ? csr 0x00000 flags 0x103 priority 3

CONFIG — SUN-3x SYSTEM

device zs0 at obio ? csr 0x62002000 flags 3 priority 3
device zs1 at obio ? csr 0x62000000 flags 0x103 priority 3

CONFIG — SUN-4 SYSTEM

device zs1 at obio ? csr 0xf0000000 flags 0x103 priority 3
device zs2 at obio 3 csr 0xe0000000 flags 3 priority 3

CONFIG — SPARCSTATION 1 SYSTEM

device-driver zs

CONFIG — Sun386i SYSTEM

device zs0 at obmem ? csr 0xFC000000 flags 3 irq 9 priority 6
device zs1 at obmem ? csr 0xA0000020 flags 0x103 irq 9 priority 6

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*n*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION

The Zilog 8530 provides 2 serial communication ports with full modem control in asynchronous mode. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the zs driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Of the synopsis lines above, the line for **zs0** specifies the serial I/O port(s) provided by the CPU board, the line for **zs1** specifies the Video Board ports (which are used for keyboard and mouse), the lines for **zs2** and **zs3** specify the first and second ports on the first SCSI board in a system, and those for **zs4** and **zs5** specify the first and second ports provided by the second SCSI board in a system, respectively.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x2** in the specification of **zs0** would treat line **/dev/ttyb** in this way.

Minor device numbers in the range 0 – 11 correspond directly to the normal tty lines and are named **/dev/ttya** and **/dev/ttyb** for the two serial ports on the CPU board and **/dev/ttys*n*** for the ports on the SCSI boards; *n* is 0 or 1 for the ports on the first SCSI board, and 2 or 3 for the ports on the second SCSI board.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 139 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 11 and is conventionally renamed **/dev/ttyd*n***, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua*n***, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line can not be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua***n* line can not be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttya**) and used for dial-in (by enabling the line for login in **/etc/ttytab**) and also used for dial-out (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

IOCTLS

The standard set of **termio ioctl( )** calls are supported by **zs**.

If the **CRTSCTS** flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the **CRTSCTS** flag is clear, the state of CTS has no effect. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl**( ) calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An **open**( ) will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl**( ) call.
EINTR	The open was interrupted by the delivery of a signal.

FILES

/dev/tty{a,b,s[0-3]}	hardwired tty lines
/dev/ttyd[0-9a-f]	dial-in tty lines
/dev/cua[0-9a-f]	dial-out tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mcp**(4S), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **ttysoftcar**(8)

DIAGNOSTICS

zs*n c* **: silo overflow.**

The 8530 character input silo overflowed before it could be serviced.

zs*n c* **: ring buffer overflow.**

The driver's character input ring buffer overflowed before it could be serviced.

Notes

NAME

intro – file formats used or read by various programs

DESCRIPTION

This section describes formats of files used by various programs.

A 5V section number means one or more of the following:

- The man page documents System V formats only.
- The man page documents default SunOS formats, and System V formats as they differ from the default formats. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents formats compliant with *IEEE Std 1003.1-1988* (POSIX.1).

LIST OF FILE FORMATS

Name	Appears on Page	Description
acct	**acct**(5)	execution accounting file
addresses	**aliases**(5)	addresses and aliases for sendmail
aliases	**aliases**(5)	addresses and aliases for sendmail
a.out	**a.out**(5)	assembler and link editor output format
ar	**ar**(5)	archive (library) file format
audit_control	**audit_control**(5)	control information for system audit daemon
audit_data	**audit_data**(5)	current information on audit daemon
audit.log	**audit.log**(5)	the security audit trail file
auto.home	**auto.home**(5)	automount map for home directories
auto.vol	**auto.vol**(5)	automount map for volumes
bar	**bar**(5)	tape archive file format
boards.pc	**boards.pc**(5)	AT- and XT-compatible boards for DOS windows
bootparams	**bootparams**(5)	boot parameter data base
bootservers	**bootservers**(5)	NIS bootservers file
coff	**coff**(5)	common assembler and link editor output
core	**core**(5)	format of memory image file
cpio	**cpio**(5)	format of cpio archive
crontab	**crontab**(5)	table of times to run periodic jobs
dir	**dir**(5)	format of directories
dump	**dump**(5)	incremental dump format
dumpdates	**dump**(5)	incremental dump format
environ	**environ**(5V)	user environment
ethers	**ethers**(5)	Ethernet address to hostname database or NIS domain
exports	**exports**(5)	directories to export to NFS clients
ext_ports	**ext_ports**(5)	external ports file for network printers, terminals, and modems
fbtab	**fbtab**(5)	framebuffer table
fcntl	**fcntl**(5)	file control options
forward	**aliases**(5)	addresses and aliases for sendmail
fs	**fs**(5)	format of a 4.2 (ufs) file system volume
fspec	**fspec**(5)	format specification in text files
fstab	**fstab**(5)	static filesystem mounting table, mounted filesystems table
ftpusers	**ftpusers**(5)	list of users prohibited by FTP
gettytab	**gettytab**(5)	terminal configuration data base
group	**group**(5)	group file
group.adjunct	**group.adjunct**(5)	group security data file
help	**help**(5)	help file format
help_viewer	**help_viewer**(5)	help viewer file format
hosts	**hosts**(5)	host name data base
hosts.equiv	**hosts.equiv**(5)	trusted hosts by system and by user

indent.pro	**indent.pro**(5)	default options for indent
inetd.conf	**inetd.conf**(5)	Internet servers database
inode	**fs**(5)	format of a 4.2 (ufs) file system volume
internat	**internat**(5)	key mapping table for internationalization
ipalloc.netrange	**ipalloc.netrange**(5)	range of addresses to allocate
keytables	**keytables**(5)	keyboard table descriptions for loadkeys and dumpkeys
lastlog	**utmp**(5V)	login records
link	**link**(5)	link editor interfaces
locale	**locale**(5)	locale database
magic	**magic**(5)	file command's magic number file
mtab	**fstab**(5)	static filesystem mounting table, mounted filesystems table
mtab	**mtab**(5)	mounted file system table
netgroup	**netgroup**(5)	list of network groups
netmasks	**netmasks**(5)	network mask data base
netrc	**netrc**(5)	file for ftp remote login data
networks	**networks**(5)	network name data base
orgrc	**orgrc**(5)	organizer configuration and initialization file
passwd	**passwd**(5)	password file
passwd.adjunct	**passwd.adjunct**(5)	user security data file
phones	**phones**(5)	remote host phone number data base
plot	**plot**(5)	graphics interface
pnp.sysnames	**pnp.sysnames**(5)	file used to allocate system names
policies	**policies**(5)	network administration policies
printcap	**printcap**(5)	printer capability data base
proto	**proto**(5)	prototype job file for at
protocols	**protocols**(5)	protocol name data base
publickey	**publickey**(5)	public key database
queuedefs	**queuedefs**(5)	queue description file for at, batch, and cron
rasterfile	**rasterfile**(5)	Sun's file format for raster images
remote	**remote**(5)	remote host description file
resolv.conf	**resolv.conf**(5)	configuration file for domain name system resolver
rfmaster	**rfmaster**(5)	Remote File Sharing name server master file
rgb	**rgb**(5)	available colors (by name) for coloredit
rhosts	**hosts.equiv**(5)	trusted hosts by system and by user
rootmenu	**rootmenu**(5)	root menu specification for SunView
rpc	**rpc**(5)	rpc program number data base
sccsfile	**sccsfile**(5)	format of an SCCS history file
services	**services**(5)	Internet services and aliases
setup.pc	**setup.pc**(5)	master configuration file for DOS
sm	**sm**(5)	in.statd directory and file structures
sm	**statmon**(5)	statd directories and file structures
sm.bak	**sm**(5)	in.statd directory and file structures
sm.bak	**statmon**(5)	statd directories and file structures
sm.state	**sm**(5)	in.statd directory and file structures
state	**statmon**(5)	statd directories and file structures
sunview	**sunview**(5)	initialization file for SunView
svdtab	**svdtab**(5)	SunView device table
syslog.conf	**syslog.conf**(5)	configuration file for syslogd system log daemon
systems	**systems**(5)	NIS systems file
tar	**tar**(5)	tape archive file format
termcap	**termcap**(5)	terminal capability data base
term	**term**(5)	terminal driving tables for nroff
term	**term**(5V)	format of compiled term file

terminfo	**terminfo**(5V)	terminal capability data base
toc	**toc**(5)	table of contents of optional clusters
translate	**translate**(5)	input and output files for system message translation
ttys	**ttytab**(5)	terminal initialization data
ttytab	**ttytab**(5)	terminal initialization data
types	**types**(5)	primitive system data types
tzfile	**tzfile**(5)	time zone information
ugid_alloc.range	**ugid_alloc.range**(5)	range of user IDs and group IDs to allocate
updaters	**updaters**(5)	configuration file for NIS updating
utmp	**utmp**(5V)	login records
uuencode	**uuencode**(5)	format of an encoded uuencode file
vfont	**vfont**(5)	font formats
vgrindefs	**vgrindefs**(5)	vgrind's language definition data base
wtmp	**utmp**(5V)	login records
xtab	**exports**(5)	directories to export to NFS clients
ypaliases	**ypaliases**(5)	NIS aliases for sendmail
ypfiles	**ypfiles**(5)	NIS database and directory structure
ypgroup	**ypgroup**(5)	NIS group file
yppasswd	**yppasswd**(5)	NIS password file
ypprintcap	**ypprintcap**(5)	NIS printer capability database

NAME

a.out – assembler and link editor output format

SYNOPSIS

#include <a.out.h>
#include <stab.h>
#include <nlist.h>

AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only. For Sun386i systems refer to **coff**(5).

DESCRIPTION

a.out is the output format of the assembler **as**(1) and the link editor **ld**(1). The link editor makes **a.out** executable files.

A file in **a.out** format consists of: a header, the program text, program data, text and data relocation information, a symbol table, and a string table (in that order). In the header, the sizes of each section are given in bytes. The last three sections may be absent if the program was loaded with the **–s** option of **ld** or if the symbols and relocation have been removed by **strip**(1).

The machine type in the header indicates the type of hardware on which the object code can be executed. Sun-2 code runs on Sun-3 systems, but not vice versa. Program files predating release 3.0 are recognized by a machine type of '**0**'. Sun-4 code may not be run on Sun-2 or Sun-3, nor vice versa.

Header

The header consists of a **exec** structure. The **exec** structure has the form:

```
struct exec {
        unsigned char   a_dynamic:1;   /* has a __DYNAMIC */
        unsigned char   a_toolversion:7; /* version of toolset used to create this file */
        unsigned char   a_machtype;    /* machine type */
        unsigned short  a_magic;       /* magic number */
        unsigned long   a_text;        /* size of text segment */
        unsigned long   a_data;        /* size of initialized data */
        unsigned long   a_bss;         /* size of uninitialized data */
        unsigned long   a_syms;        /* size of symbol table */
        unsigned long   a_entry;       /* entry point */
        unsigned long   a_trsize;      /* size of text relocation */
        unsigned long   a_drsize;      /* size of data relocation */
};
```

The members of the structure are:

a_dynamic 1 if the **a.out** file is dynamically linked or is a shared object, **0** otherwise.

a_toolversion The version number of the toolset (**as**, **ld**, etc.) used to create the file.

a_machtype One of the following:

- **0** pre-3.0 executable image
- **M_68010** executable image using only MC68010 instructions that can run on Sun-2 or Sun-3 systems.
- **M_68020** executable image using MC68020 instructions that can run only on Sun-3 systems.
- **M_SPARC** executable image using SPARC instructions that can run only on Sun-4 systems.

a_magic One of the following:

- **OMAGIC** An text executable image which is not to be write-protected, so the data segment is immediately contiguous with the text segment.

NMAGIC A write-protected text executable image. The data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. When the image is started with **execve**(2V), the entire text and data segments will be read into memory.

ZMAGIC A page-aligned text executable image. the data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. The text and data sizes are both multiples of the page size, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is the default format produced by **ld**(1).

The macro **N_BADMAG** takes an **exec** structure as an argument; it evaluates to 1 if the **a_magic** field of that structure is invalid, and evaluates to 0 if it is valid.

a_text The size of the text segment, in bytes.

a_data The size of the initialized portion of the data segment, in bytes.

a_bss The size of the "uninitialized" portion of the data segment, in bytes. This portion is actually initialized to zero. The zeroes are not stored in the **a.out** file; the data in this portion of the data segment is zeroed out when it is loaded.

a_syms The size of the symbol table, in bytes.

a_entry The virtual address of the entry point of the program; when the image is started with **execve**, the first instruction executed in the image is at this address.

a_trsize The size of the relocation information for the text segment.

a_drsize The size of the relocation information for the data segment.

The macros **N_TXTADDR**, **N_DATADDR**, and **N_BSSADDR** give the memory addresses at which the text, data, and bss segments, respectively, will be loaded.

In the **ZMAGIC** format, the size of the header is included in the size of the text section; in other formats, it is not.

When an **a.out** file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized data), and a stack. For the **ZMAGIC** format, the header is loaded with the text segment; for other formats it is not.

Program execution begins at the address given by the value of the **a_entry** field.

The stack starts at the highest possible location in the memory image, and grows downwards. The stack is automatically extended as required. The data segment is extended as requested by **brk**(2) or **sbrk**.

Text and Data Segments

The text segment begins at the start of the file for **ZMAGIC** format, or just after the header for the other formats. The **N_TXTOFF** macro returns this absolute file position when given an **exec** structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The **N_DATOFF** macro returns the absolute file position of the beginning of the data segment when given an **exec** structure as argument.

Relocation

The relocation information appears after the text and data segments. The **N_TRELOFF** macro returns the absolute file position of the relocation information for the text segment, when given an **exec** structure as argument. The **N_DRELOFF** macro returns the absolute file position of the relocation information for the data segment, when given an **exec** structure as argument. There is no relocation information if **a_trsize+a_drsize==0**.

Relocation (Sun-2 and Sun-3 Systems)

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to the bytes in the file. If a byte involves a reference to a relative location, or relocatable segment, then the value stored in the file is an offset from the associated segment.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
struct reloc_info_68k {
     long      r_address;           /* address which is relocated */
unsigned int  r_symbolnum:24,      /* local symbol ordinal */
     r_pcrel:1,           /* was relocated pc relative already */
     r_length:2,          /* 0=byte, 1=word, 2=long */
     r_extern:1,          /* does not include value of sym referenced */
     r_baserel:1,         /* linkage table relative */
     r_jmptable:1,        /* pc-relative to jump table */
     r_relative:1,        /* relative relocation */
     :1;
};
```

If **r_extern** is 0, then **r_symbolnum** is actually an **n_type** for the relocation (for instance, **N_TEXT** meaning relative to segment text origin.)

Relocation (Sun-4 System)

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is ignored. Unlike the Sun-2 and Sun-3 system, the offset from the associated symbol is kept with the relocation record. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to this offset, and the sum is inserted into the bytes in the text or data segment.

If relocation information is present, it amounts to twelve bytes per relocatable datum as in the following structure:

```
enum reloc_type
{
    RELOC_8,            RELOC_16,           RELOC_32,           /* simplest relocs */
    RELOC_DISP8,        RELOC_DISP16,       RELOC_DISP32,       /* Disp's (pc-rel) */
    RELOC_WDISP30,      RELOC_WDISP22,      /* SR word disp's */
    RELOC_HI22,         RELOC_22,           /* SR 22-bit relocs */
    RELOC_13,           RELOC_LO10,         /* SR 13&10-bit relocs */
    RELOC_SFA_BASE,     RELOC_SFA_OFF13,    /* SR S.F.A. relocs */
    RELOC_BASE10,       RELOC_BASE13,       RELOC_BASE22,       /* base_relative pic */
    RELOC_PC10,         RELOC_PC22,         /* special pc-rel pic*/
    RELOC_JMP_TBL,      /* jmp_tbl_rel in pic */
    RELOC_SEGOFF16,     /* ShLib offset-in-seg */
    RELOC_GLOB_DAT,     RELOC_JMP_SLOT,     RELOC_RELATIVE,     /* rtld relocs */
};

struct reloc_info_sparc   /* used when header.a_machtype == M_SPARC */
{
    unsigned long int   r_address;          /* relocation addr (offset in segment) */
    unsigned int        r_index     :24;    /* segment index or symbol index */
    unsigned int        r_extern    : 1;    /* if F, r_index==SEG#; if T, SYM idx */
    int                 : 2;                /* <unused> */
    enum reloc_type     r_type      : 5;    /* type of relocation to perform */
    long int            r_addend;           /* addend for relocation value */
};
```

If **r_extern** is 0, then **r_index** is actually a **n_type** for the relocation (for instance, **N_TEXT** meaning relative to segment text origin.)

Symbol Table

The **N_SYMOFF** macro returns the absolute file position of the symbol table when given an **exec** structure as argument. Within this symbol table, distinct symbols point to disjoint areas in the string table (even when two symbols have the same name). The string table immediately follows the symbol table; the **N_STROFF** macro returns the absolute file position of the string table when given an **exec** structure as argument. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table. This size *includes* the 4 bytes; thus, the minimum string table size is 4. Layout information as given in the include file for the Sun system is shown below.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
struct nlist {
        union {
                char    *n_name;        /* for use when in-memory */
                long    n_strx;         /* index into file string table */
        } n_un;
        unsigned char   n_type;         /* type flag, that is, N_TEXT etc; see below */
        char    n_other;
        short   n_desc;                 /* see <stab.h> */
        unsigned        n_value;        /* value of this symbol (or adb offset) */
};
#define n_hash  n_desc                  /* used internally by ld */
/*
* Simple values for n_type.
*/
#define N_UNDF  0x0                     /* undefined */
#define N_ABS   0x2                     /* absolute */
#define N_TEXT  0x4                     /* text */
#define N_DATA  0x6                     /* data */
#define N_BSS   0x8                     /* bss */
#define N_COMM  0x12                    /* common (internal to ld) */
#define N_FN    0x1f                    /* file name symbol */
#define N_EXT   01                      /* external bit, or'ed in */
#define N_TYPE  0x1e                    /* mask for all the type bits */

/*
* Other permanent symbol table entries have some of the N_STAB bits set.
* These are given in <stab.h>
*/
#define N_STAB  0xe0                    /* if any of these bits set, don't discard */
```

In the **a.out** file a symbol's **n_un.n_strx** field gives an index into the string table. A **n_strx** value of 0 indicates that no name is associated with a particular symbol table entry. The field **n_un.n_name** can be used to refer to the symbol name only if the program sets this up using **n_strx** and appropriate data from the string table. Because of the union in the **nlist** declaration, it is impossible in C to statically initialize such a structure. If this must be done (as when using **nlist**(3V)) include the file **<nlist.h>**, rather than **<a.out.h>**. This contains the declaration without the union.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader **ld** as the name of a common region whose size is indicated by the value of the symbol.

SEE ALSO

adb(1), **as**(1), **cc**(1V), **dbx**(1), **ld**(1), **nm**(1), **strip**(1), **brk**(2), **nlist**(3V), **coff**(5)

NAME

acct – execution accounting file

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

The **acct**(2V) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
typedef u_short comp_t;

struct acct
{
        char    ac_flag;        /* Accounting flag */
        char    ac_stat;        /* Exit status */
        uid_t   ac_uid;                 /* Accounting user ID */
        gid_t   ac_gid;                 /* Accounting group ID */
        dev_t   ac_tty;                 /* control typewriter */
        time_t  ac_btime;               /* Beginning time */
        comp_t  ac_utime;               /* Accounting user time */
        comp_t  ac_stime;               /* Accounting system time */
        comp_t  ac_etime;               /* Accounting elapsed time */
        comp_t  ac_mem;                         /* average memory usage */
        comp_t  ac_io;                  /* chars transferred */
        comp_t  ac_rw;                  /* blocks read or written */
        char    ac_comm[8];             /* Accounting command name */
};
```

The type **comp_t** is a 3 bits base 8 exponent, 13 bit fraction "floating point" number. If the process does an **execve**(2V), the first 8 characters of the filename appear in **ac_comm**. **ac_flag** contains bits indicating whether **execve**(2V) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2V), **execve**(2V), **sa**(8)

NAME

aliases, addresses, forward – addresses and aliases for sendmail

SYNOPSIS

/etc/aliases
/etc/aliases.dir
/etc/aliases.pag
~/.forward

DESCRIPTION

These files contain mail addresses or aliases, recognized by **sendmail**(8), for the local host:

/etc/passwd	Mail addresses (usernames) of local users.
/etc/aliases	Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.
/etc/aliases.{dir,pag}	The aliasing information from **/etc/aliases**, in binary, **dbm**(3X) format for use by **sendmail**(8). The program **newaliases**(8), which is invoked automatically by **sendmail**(8), maintains these files.
~/.forward	Addresses to which a user's mail is forwarded (see **Automatic Forwarding**, below).

In addition, the Network Information Service (NIS) aliases map *mail.aliases* contains addresses and aliases available for use across the network.

Addresses

As distributed, **sendmail**(8) supports the following types of addresses:

Local Usernames

username

Each local *username* is listed in the local host's **/etc/passwd** file.

Local Filenames

pathname

Messages addressed to the absolute *pathname* of a file are appended to that file.

Commands

|*command*

If the first character of the address is a vertical bar, (|), **sendmail**(8) pipes the message to the standard input of the *command* the bar precedes.

TCP/IP-*standard Addresses*

username@*domain*

If *domain* does not contain any '.' (dots), then it is interpreted as the name of a host in the current domain. Otherwise, the message is passed to a *mailhost* that determines how to get to the specified domain. Domains are divided into subdomains separated by dots, with the top-level domain on the right. Top-level domains include:

.COM	Commercial organizations.
.EDU	Educational organizations.
.GOV	Government organizations.
.MIL	Military organizations.

For example, the full address of John Smith could be:

js@jsmachine.Podunk-U.EDU

if he uses the machine named **jsmachine** at Podunk University.

uucp(1C) *Addresses*

... [*host*!]*host*!*username*

These are sometimes mistakenly referred to as "Usenet" addresses. **uucp**(1C) provides links to numerous sites throughout the world for the remote copying of files.

Other site-specific forms of addressing can be added by customizing the **sendmail** configuration file. See the **sendmail**(8), and *System and Network Administration* for details. Standard addresses are recommended.

Aliases

Local Aliases

/etc/aliases is formatted as a series of lines of the form

aliasname: *address* [, *address*]

aliasname is the name of the alias or alias group, and *address* is the address of a recipient in the group. Aliases can be nested. That is, an *address* can be the name of another alias group. Because of the way **sendmail** performs mapping from upper-case to lower-case, an *address* that is the name of another alias group must not contain any upper-case letters.

Lines beginning with white space are treated as continuation lines for the preceding alias. Lines beginning with # are comments.

Special Aliases

An alias of the form:

owner–*aliasname*: *address*

directs error-messages resulting from mail to *aliasname* to *address*, instead of back to the person who sent the message.

An alias of the form:

aliasname: **:include:***pathname*

with colons as shown, adds the recipients listed in the file *pathname* to the *aliasname* alias. This allows a private list to be maintained separately from the aliases file.

NIS Domain Aliases

Normally, the aliases file on the master NIS server is used for the *mail.aliases* NIS map, which can be made available to every NIS client. Thus, the **/etc/aliases*** files on the various hosts in a network will one day be obsolete. Domain-wide aliases should ultimately be resolved into usernames on specific hosts. For example, if the following were in the domain-wide alias file:

jsmith:js@jsmachine

then any NIS client could just mail to **jsmith** and not have to remember the machine and username for John Smith. If an NIS alias does not resolve to an address with a specific host, then the name of the NIS domain is used. There should be an alias of the domain name for a host in this case. For example, the alias:

jsmith:root

sends mail on an NIS client to **root@podunk-u** if the name of the NIS domain is **podunk-u**.

Automatic Forwarding

When an alias (or address) is resolved to the name of a user on the local host, **sendmail** checks for a **.forward** file, owned by the intended recipient, in that user's home directory, and with universal read access. This file can contain one or more addresses or aliases as described above, each of which is sent a copy of the user's mail.

Care must be taken to avoid creating addressing loops in the **.forward** file. When forwarding mail between machines, be sure that the destination machine does not return the mail to the sender through the operation of any NIS aliases. Otherwise, copies of the message may "bounce". Usually, the solution is to change the NIS alias to direct mail to the proper destination.

A backslash before a username inhibits further aliasing. For instance, to invoke the **vacation**(1) program, user **js** creates a **.forward** file that contains the line:

\js, "|/usr/ucb/vacation js"

so that one copy of the message is sent to the user, and another is piped into the **vacation**(1) program.

FILES

/etc/passwd
/etc/aliases
~/.forward

SEE ALSO

uucp(1C), **vacation**(1), **dbm**(3X), **newaliases**(8), **sendmail**(8)

System and Network Administration

BUGS

Because of restrictions in **dbm**(3X) a single alias cannot contain more than about 1000 characters. Nested aliases can be used to circumvent this limit.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

ar – archive (library) file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command **ar** combines several files into one. Archives are used mainly as libraries to be searched by the link-editor **ld**(1).

A file produced by **ar** has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG   "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
```

The name is a blank-padded string. The **ar_fmag** field contains **ARFMAG** to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for **ar_mode**, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a NEWLINE is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Sun386i DESCRIPTION

The file produced by *ar* on Sun386i systems is identical to that described above with the following changes:

Each archive containing COFF files [see **coff**(5)] includes an archive symbol table. This symbol table is used by the link editor **ld** to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by **ar**.

The *ar_name* field of the ar_hdr structure described above is blank-padded and slash (/) terminated. Common format archives can be moved from system to system as long as the portable archive command **ar** is used. Conversion tools such as **convert** exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on an even byte boundary; a NEWLINE is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0]** == '/'). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".

- The name string table. Length: *ar_size* – (4 bytes * ("the number of symbols" + 1)).

The number of symbols and the array of offsets are managed with *sgetl* and *sputl*. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO

ar(1V), **ld**(1), **nm**(1)

Sun386i WARNINGS

strip(1) will remove all archive symbol entries from the header. The archive symbol entries must be restored via the **ts** option of the *ar*(1V) command before the archive can be used with the link editor *ld*(1).

BUGS

Filenames lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

audit.log – the security audit trail file

SYNOPSIS

#include <sys/label.h>
#include <sys/audit.h>
#include <sys/user.h>

DESCRIPTION

The **audit.log** file begins with a header record consisting of an **audit_header** structure followed by the previous audit file name. When the audit daemon is started (usually only at boot time), the previous audit file name is NULL.

```
struct audit_header {
        int      ah_magic;        /* magic number */
        time_t   ah_time;         /* the time */
        short    ah_namelen;      /* length of file name */
};
typedef struct audit_header audit_header_t;
```

The file may end with a trailer record consisting of an **audit_trailer** structure followed by the name of the next audit file.

```
struct audit_trailer {
        short    at_record_size;        /* size of this */
        short    at_record_type;        /* its type, a trailer */
        time_t   at_time;               /* the time */
        short    at_namelen;            /* length of file name */
};
typedef struct audit_trailer audit_trailer_t;
```

The **audit.log** file contains audit records in their raw form. The records are of varying size depending on the record type. Each record has a header which is an **audit_record** structure.

```
struct audit_record {
        short        au_record_size;     /* size of this */
        short        au_record_type;     /* its type */
        time_t       au_time;            /* the time */
        short        au_uid;             /* real uid */
        short        au_auid;            /* audit uid */
        short        au_euid;            /* effective */
        short        au_gid;             /* real group */
        short        au_pid;             /* effective */
        int          au_errno;           /* error code */
        int          au_return;          /* a return value */
        blabel_t     au_label;           /* also ... */
        short        au_param_count;             /* # of parameters */
};
typedef struct audit_record audit_record_t;
```

Immediately following the header is a set of two byte integers, the number of which exist for a given record is contained in the **au_param_count** field. These numbers are the lengths of the additional data items. The additional data items follow the list of lengths, the first length describing the first data item. Interpretation of this data is left to the program accessing it.

SEE ALSO

audit(2), **audit**(8)

Security Features Guide

NAME

audit_control – control information for system audit daemon

SYNOPSIS

/etc/security/audit/audit_control

DESCRIPTION

The **audit_control** file contains audit control information read by **auditd**(8). Each line consists of a title and a string, separated by a colon. There are no restrictions on the order of lines in the file, although some lines must appear only once. A line beginning with '#' is a comment.

Directory definition lines list the directories to be used when creating audit files, in the order in which they are to be used. The format of a directory line is:

dir: *directory-name*

where *directory-name* is the name of a directory in which to create audit files, with the form:

/etc/security/audit/*server***/***machine*

where *server* is the name of an audit file system on the machine where this audit directory resides, and *machine* is the name of the local machine, since audit files belonging to different machines are, by convention, stored in separate subdirectories of a single audit directory. The naming convention normally has *server* be the name of a server machine, and all clients mount **/etc/security/audit/***server* at the same location in their local file systems. If the same server exports several different file systems for auditing, their *server* names will, of course, be different.

The audit threshold line specifies the percentage of free space that must be present in the file system containing the current audit file. The format of the threshold line is:

minfree: *percentage*

where *percentage* is indicates the amount of free space required. If free space falls below this threshold, the audit daemon **auditd**(8) invokes the shell script **/etc/security/audit/audit_warn**. If no threshold is specified, the default is 0%.

The audit flags line specifies the default system audit value. This value is combined with the user audit value read from **/etc/security/passwd.adjunct** to form the process audit state. The user audit value overrides the system audit value. The format of a flags line is:

flags: *audit-flags*

where *audit-flags* specifies which event classes are to be audited. The character string representation of *audit-flags* contains a series of flag names, each one identifying a single audit class, separated by commas. A name preceded by '–' means that the class should be audited for failure only; successful attempts are not audited. A name preceded by '+' means that the class should be audited for success only; failing attempts are not audited. Without a prefix, the name indicates that the class is to be audited for both successes and failures. The special string **all** indicates that all events should be audited; –**all** indicates that all failed attempts are to be audited, and +**all** all successful attempts. The prefixes ^, ^–, and ^+ turn off flags specified earlier in the string (^– and ^+ for failing and successful attempts, ^ for both). They are typically used to reset flags.

The following table lists the audit classes:

short name	long name	short description
dr	**data_read**	Read of data, open for reading, etc.
dw	**data_write**	Write or modification of data
dc	**data_create**	Creation or deletion of any object
da	**data_access_change**	Change in object access (modes, owner)
lo	**login_logout**	Login, logout, creation by **at**(1)
ad	**administrative**	Normal administrative operation
p0	**minor_privilege**	Privileged operation
p1	**major_privilege**	Unusual privileged operation

EXAMPLE

Here is a sample **/etc/security/audit_control** file for the machine eggplant:

```
dir: /etc/security/audit/jedgar/eggplant
dir: /etc/security/audit/jedgar.aux/eggplant
#
# Last-ditch audit file system when jedgar fills up.
#
dir: /etc/security/audit/global/eggplant
minfree: 20
flags: lo,p0,p1,ad,-all,^-da
```

This identifies server **jedgar** with two file systems normally used for audit data, another server **global** used only when **jedgar** fills up or breaks, and specifies that the warning script is run when the file systems are 80% filled. It also specifies that all logins, privileged and administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to access data are to be audited.

FILES

/etc/security/audit/audit_control
/etc/security/audit/audit_warn
/etc/security/audit/*/*/*
/etc/security/passwd_adjunct

SEE ALSO

at(1), **audit**(2), **getfauditflags**(3), **audit.log**(5), **audit**(8), **auditd**(8)

NAME

audit_data – current information on audit daemon

SYNOPSIS

/etc/security/audit/audit_data

DESCRIPTION

The **audit_data** file contains information about the audit daemon. The file contains the process ID of the audit daemon, and the pathname of the current audit log file. The format of the file is:

<pid>:*<pathname>*

Where *pid* is the process ID for the audit daemon, and *pathname* is the full pathname for the current audit log file.

EXAMPLE

64:/etc/security/audit/auditserv/auditclient/2df0504

FILES

/etc/security/audit/audit_data

SEE ALSO

audit(2), **audit.log**(5), **audit**(8), **auditd**(8)

NAME

auto.home – automount map for home directories

SYNOPSIS

/etc/auto.home

AVAILABILTITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

auto.home resides in the **/etc** directory, and contains **automount**(8) map entries for user's home directories. On Sun386i systems, this file is used to build the **auto.home** Network Information Service (NIS) map used by **automount** at system startup and reads the **auto.master** NIS database, which contains an entry for **auto.home** and **/home .** The **auto.home** map contains entries for each username in the NIS **passwd** map, and the *hostname:/directory* to NFS mount.

References to **/home/***username* are translated by the automount daemon using the **auto.home** map, and the directory specified in the map entry is **nfs** mounted and that directory returned to the user's program.

User accounts created using **snap**(1) or **logintool**(8) have **passwd**(5) entries where the initial (home) directory name is, in the form **/home/***username*. **snap** and **logintool** also automatically create the **auto.home** entry for a user account. The format of the entry is described in **automount**(8). An example entry is:

mtravis **system2:/export/home/users/mtravis**

Thus, when the user **mtravis** logs into a Sun386i systems, the automounter automatically mounts his home directory from **system2**. This allows a user to log in to any Sun386i workstation on the network and be automatically placed in their home directory.

The convention for the format of home directory names used by **snap** and **logintool** is:

/export/home/*groupname/username*

Note: this is a different map and mechanism for home directories than the one that the automount daemon provides with the **–homes** switch. This is because the Sun386i convention for the format of home directory names differs and provides directories that can be used as mount points on a per user and per group basis.

FILES

/etc/auto.home

SEE ALSO

snap(1), **passwd**(5), **automount**(8), **logintool**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

auto.vol – automount map for volumes

SYNOPSIS

/etc/auto.vol

AVAILABILTITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

auto.vol resides in the **/etc** directory, and contains **automount**(8) map entries for **volumes**. On Sun386i systems, this file is used to build the **auto.vol** Network Information Service (NIS) map used by **automount**(8) at system startup. **automouunt** reads the **auto.master** NIS map, which contains an entry for **auto.vol** and **/vol.**

References to **/vol/***volume_name* are translated by the automount daemon using the **auto.vol** map, and the directory specified in the map entry is mounted.

The concept of a volume is that it is a self contained directory hierarchy that can be NFS mounted. It is referenced using a known *volume_name*. The use of an automount map is suggested so that the volume and its contents can be referenced through **/vol**. This is advantageous because location-transparency (that is, which host the volume is on) and replication of read-only volumes can be provided using the automount mechanism. The format of the entry is described in **automount**(8). An example entry is:

archive **system4:/export/archive**

In the above example, the **archive** volume is currently on line on **system4**. Users and programs can reference it via **/vol/archive**. If for some reason the volume had to be moved to another system, **system2** for example, the network or system administrator simply edits the map entry for the archive volume and changes the hostname to **system2** and then rebuilds the NIS maps.

archive **system2:/export/archive**

Users and programs can continue to refer to the archive volume using **/vol/archive**, unaware that the volume was moved to another system.

FILES

/etc/auto.vol

SEE ALSO

automount(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

bar – tape archive file format

DESCRIPTION

bar(1), (the tape archive command) dumps several files into one, in a medium suitable for transportation. This format is not compatible with the format generated by **tar**(1).

A *bar tape* or file is a series of blocks. Each block is of size **TBLOCK.** A file on the tape is represented by a header block that describes the file, followed by zero or more blocks that give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an EOF indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **bar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK512

union hblock {
        char dummy[TBLOCK];
        struct header {
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char rdev[8];
                char linkflag;
                char bar_magic[2];
                char volume_num[4];
                char compressed;
                char date[12];
                char start_of_name;
        } dbuf;
};
```

start_of_name is a null-terminated string. *date* is the date of the archive. *bar_magic* is a special number indicating that this is a **bar** archive. *rdev* is the device type, for files that are devices. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except *size*, *rdev*, and *mtime*, which do not contain the trailing null. *start_of_name* is the name of the file, as specified on the *bar* command line. Files dumped because they were in a directory that was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers that own the file. *size* is the size of the file in bytes. Links and symbolic links, and special files, are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value that represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII 0 if the file is "normal" or a special file, 1 if it is an hard link, 2 if it is a symbolic link, and 3 if it is a special file (device or FIFO). The name linked-to, if any, is in a null-terminated string, following *start_of_name*. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

When the **H** modifier is used with **bar** , an additional header block (one that does not pertain to a particular file) is written to the first block of each volume of the archive. The header ID, as specified on the command line, is copied to *start_of_name*. The size reflects the number of bytes to skip to the start of the first full file (always zero on the first volume).

The encoding of the header is designed to be portable across machines.

SEE ALSO

bar(1)

NAME

boards.pc – information about AT- and XT-compatible boards for DOS windows

SYNOPSIS

/etc/dos/defaults/boards.pc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **boards.pc** file stores information about AT- and XT-compatible boards installed on a system.

Only the super-user may alter the file.

The file format is as follows, with entries separated by SPACE or TAB characters:

Board-name I/O port range IRQ DMA Memory Options

Board-name
The name of the board as it will appear in the DOS Windows Device menu. Use any name that is not longer than 19 characters.

I/O port range
Most boards have I/O addresses through which they exchange information with the workstation. For boards that will be used by DOS, the I/O address is entered in the **boards.pc** file, directly to the right of the board name.

Certain I/O addresses are already used by DOS Windows emulated devices (such as drive C and the DOS printers), and by built-in system hardware. The following list shows the AT-bus I/O address spaces:

Address	**DOS Use**
1F8-1FF *	**Hard disk (C:) emulation**
218-21F	**Expanded memory**
230-23F	**Bus mouse emulation**
278-27F	**Parallel port 2 (usually accessed through LPT3)**
378-37F *	**Parallel port 1 (usually accessed through LPT2)**
3B0-3BF	**Monochrome display adapter**
3D0-3DF	**Color display adapter**
3F0-3F7 *	**Diskette controller**

An address marked with an asterisk cannot be replaced by a board. When the board you are installing uses one of these addresses, or it uses the same address as another board that is already installed, change the jumpers or switch settings on your board to use a different address. If you add a board that occupies one of these address spaces, DOS ignores the entry. An address not marked with an asterisk may be used for a board you are installing, as long as you do not plan to use the emulated device at that address.

Adding an I/O Address Entry to boards.pc:

If the board uses addresses that can be contained within one eight-address block, note the block base address and include it in the *I/O port range* column of the **boards.pc** file. When using a multiple-block address, specify the base address of each block. For example, when entering a two-block address, specify the base addresses of both the first and second blocks, and separated with a SPACE character. Suppose you have a board with a two-block I/O address space that begins at 380. You would specify 380 388 in the **boards.pc** file's *I/O port range* column.

IRQ Some boards send periodic signals asking DOS to delay whatever it is doing and accept information from the device. These signals are known as **interrupt requests**, or more simply, as **interrupts**. The following chart shows the interrupt levels available under DOS Windows. Valid interrupt levels are 1 to 15, although some of these are reserved for emulated DOS devices.

Interrupt Level	Availability
0	Unavailable; used for timer emulation
1	Unavailable; used for keyboard emulation
2	Unavailable; used for interrupt controller 2 cascade
3	Available for board, unless COM2 emulation in use (specified in setup.pc)
4	Available for board, unless COM1 emulation in use (specified in setup.pc)
5	Available for board, unless LPT3 emulation in use (specified in setup.pc)
6	Unavailable; used for diskette drive emulation
7	Unavailable; used by built-in parallel port
8	Unavailable; used for real-time clock emulation
9	Available for board
10	Available for board
11	Available for board
12	Available for board
13	Unavailable; used for 8087 numeric coprocessor emulation
14	Unavailable; used for hard disk emulation
15	Available for board

To ensure that signals do not become confused, set each board or emulated device that uses interrupts for a different interrupt level. Normally, interrupt settings are changed by pressing small switches or moving metal jumpers on the board itself. Consult the manual of the board you are installing for details on how this is done. In addition to the changes required on the board itself, make sure that the interrupt level in your **boards.pc** file matches the setting on the card. For example, if a board's physical interrupt was previously 3, and you change it to 4 by altering switch settings or board jumpers, make a corresponding change in the **boards.pc** file. If the card uses a DOS driver, you may also need to make changes in **C:CONFIG.SYS** or other files to reflect the switch settings on the card.

Adding an Interrupt Entry to boards.pc

Some boards do not generate interrupts, and therefore will not have an interrupt level listed in their manuals. If this is the case, leave the *IRQ* column empty. For boards where an interrupt level is required, enter the letters **irq** followed by the appropriate number in the **boards.pc** file, as shown in EXAMPLES below.

DMA Certain boards use direct memory access (DMA) channels to ensure speedy transfer of large quantities of data. DMA channels 0, 1, 3, and 5 are available. Each DOS or SunOS DMA board on the system must be assigned a unique DMA channel. When two or more boards expect to use DMA channel 1, physically alter DMA settings on one of the boards so that it uses a different channel (such as DMA channel 3). Normally these settings are changed by pressing small switches or moving metal jumpers on the board itself. Consult the manual for the board you are installing for details on changing a DMA channel setting.

Adding a DMA Entry to boards.pc

When the board you are installing uses a DMA channel, include a **dma** entry for that board. For example, when the board is set up to use DMA channel 3, the entry can look like this:

MYBOARD 200 208 irq 2 dma 3

Memory

Some boards are equipped with memory chips for DOS. Because this memory is "mapped" (transferred) into DOS memory so that DOS can read it, the boards are called *memory mapped boards*. When you install such a board, include a **mem** entry with the following format:

mem *address size*

The *address* is the starting address of the memory segment, in hexadecimal notation. Enter the size of the memory block in kilobytes, in decimal notation. The following example is for a board that starts mapped memory at the address $DE00 and uses a block of 8 kilobytes.

MYBOARD 258 irq 5 dma 3 mem de00 8

When determining the size of the memory block, be careful not to confuse DOS address size (the number you should use) with actual on-board memory (the number you should not use). For example, a LIM memory board might have 2 megabytes of on-board memory, yet may require only 64 kilobytes of DOS address space for its memory mapping. Therefore, the number to use for the **mem** entry is 64.

Options

reboot

Certain boards require DOS rebooting before they work. These same boards require that you reboot DOS after you have finished using them. You can set up DOS to reboot the current DOS window automatically whenever the board is attached. DOS displays a confirmatory alert before rebooting.

To force DOS to reboot when you attach the board, add the word **reboot** at the end of the **boards.pc** line for that board, as shown in the following example:

MYBOARD 3e8 mem a000 192 reboot

If you choose to omit the **reboot** instruction, you can enable the board by attaching it and then manually rebooting:

1. Choose **Attached from the Device** menu to enable the board.
2. Choose **Reboot DOS Window**.

To detach such a board from a DOS window, choose **Detach** and then reboot the DOS window.

shared

You can specify that a device is to be shared between windows, rather than being reserved for use by one window at a time. Generally, you should do this only with devices, such as joysticks, which can fluidly move from one DOS window to another. To designate a device as shared, place the word **shared** at the very end of the **boards.pc** line:

Joystick 200 shared

Determining Board Information

In many cases, you may need to determine whether a board you are installing will conflict with other devices on the system. Also, you sometimes may need to install a board for which there is no entry in the **boards.pc** file. In most cases, the instruction manual included with the board you are installing should contain the technical information you need, including:

The I/O port addresses at which the board is accessed. One or more blocks can be reserved, and there are eight consecutive addresses per block.

The board's interrupt level, if the board generates interrupts.

The DMA channel number, if the board uses a direct memory access channel.

Memory mapping information, if the board maps data into DOS memory.

If the board's manual does not provide such information, contact the manufacturer.

EXAMPLES

The following is an example of a **boards.pc** file:

#COM2	**2f8**	**irq 3**			
#Joystick	**200**				**shared**
#EGA	**3b0 3b8 3c0 3c8 3d0 3d8**			**mem a000 192**	**reboot**
#VGA	**3b0 3b8 3c0 3c8 3d0 3d8 102 2e8**			**mem a000 192**	
#3COM-3C501	**300 308**	**irq 3**	**dma 1**		
#TOPS-FlashTalk	**398**	**irq 3**			
#IBM-3363-Worm	**258**	**irq 5**	**dma 3**	**mem de00 8**	**reboot**
#Plus-Hardcard20	**320**	**irq 5**	**dma 3**	**mem ca00 8**	**reboot**
#HP-Basic	**390**	**irq 3**			
#DCA-IRMA1	**220 228**				
#DCA-IRMA2	**220 228 280 288**				
#Bernoulli-A220H	**350**				**reboot**
#WD8003E	**280 288 290 298**	**irq 5**		**mem d000 8**	
#NI5210	**360**	**irq 5**		**mem c000 16**	
#NIC		**irq 5**		**mem d000 32**	
#LPT2	**278**	**irq 5**			

FILES

/usr/lib/help/*/*

SEE ALSO

dos(1), **setup.pc**(5)

Sun386i Advanced Skills

NAME

bootparams – boot parameter data base

SYNOPSIS

/etc/bootparams

DESCRIPTION

The **bootparams** file contains the list of client entries that diskless clients use for booting. For each diskless client the entry should contain the following information:

name of client
a list of keys, names of servers, and pathnames.

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

A client entry in the local **/etc/bootparams** file supersedes an entry in the corresponding Network Information Service (NIS) map.

EXAMPLE

Here is an example of the /etc/**bootparams** taken from a SunOS system.

```
myclient        root=myserver:/nfsroot/myclient \
        swap=myserver:/nfsswap/myclient \
        dump=myserver:/nfsdump/myclient
```

FILES

/etc/bootparams

SEE ALSO

bootparamd(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

bootservers – NIS bootservers file

SYNOPSIS

/etc/bootservers

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **bootservers** file is an ASCII file that resides in the **/etc** directory on the Network Information Service (NIS) master server. The file contains basic information about each host providing boot services for clients on the network. This file contains a one-line entry for each boot server, where each field *must* be separated by a TAB character:

system type client_limit swap_size tmp_size root_minfree swap_minfree

The entries in the file have the following descriptions:

system	is the name of a boot server. This field contains only lowercase and numeric characters, must start with a lower-case character, and must not be longer than 32 characters.
type	Currently, the only legal value is 3.
client_limit	indicates the maximum number of diskless clients the server is willing to accept.
swap_size	default swap size per client (in kilobytes).
tmp_size	default tmp size per client (in kilobytes).
root_minfree	minimum amount of disk space in the server's client-root partition after a client is added (in kilobytes).
swap_minfree	minimum amount of disk space in the server's client-swap partition after a client is added (in kilobytes).

EXAMPLE

Here is a sample **bootservers** file entry:

polaris 3 2 16000 8000 40000 0

FILES

/etc/bootservers

SEE ALSO

System and Network Administration,
Sun386i Advanced Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

coff – common assembler and link editor output

SYNOPSIS

#include <filehdr.h>
#include <aouthdr.h>
#include <scnhdr.h>
#include <reloc.h>
#include <linenum.h>
#include <storclass.h>
#include <syms.h>

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The output from the link editor and the assembler (named **a.out** by default) is in COFF format (Common Object File Format) on the Sun386i system.

A common object file consists of a file header, a system header (if the file is link editor output), a table of section headers, a data section, relocation information, (optional) line numbers, a symbol table, and a string table. The general format looks like this:

file-header
system-header
section-headers
data
relocation
line-numbers
symbol-table
string-table

section-headers contains a number of section headers:

section 1 header
...
section n header

Similarly, *data*, *relocation*, and *line-numbers* are each divided into *n* sections.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **–s** option of **ld**(1) or if they were removed by **strip**(1). Also note that the relocation information will be absent after linking unless the **–r** option of **ld**(1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment starts at location 0x1000 by default.

The **a.out** file produced by **ld**(1) has the magic number 0413 in the first field of the system header. The headers (file header, system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal 0x1000 plus the size of the headers, and will vary depending upon the number of section headers in the **a.out** file. In an **a.out** file with three sections (.text, .data, .bss, and .comment), the first text address is at 0x000010D0. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes will share a single text segment.

The data segment starts at the next 4K boundary past the last text address. The first data address is determined by the following: If an **a.out** file were split into 4K chunks, one of the chunks would contain both the end of text and the beginning of data. When the **a.out** file is loaded into memory for execution, that chunk will appear twice; once at the end of text and once at the beginning of data (with some unused space in between). The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the page size without having to realign the beginning of the data section to a page boundary. Therefore the first data address is the sum of the next segment boundary past the end of text plus the remainder of the last text address divided by 4K. If the last text address is a multiple of 4K no duplication is necessary.

On the Sun386i computer the stack begins at location 0xFBFFFFFF and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk**(2) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the file header is:

```
struct filehdr
{
        unsigned short  f_magic;    /* magic number */
        unsigned short  f_nscns;    /* number of sections */
        long            f_timdat;   /* time and date stamp */
        long            f_symptr;   /* file ptr to symtab */
        long            f_nsyms;    /* # symtab entries */
        unsigned short  f_opthdr;   /* sizeof(opt hdr) */
        unsigned short  f_flags;    /* flags */
};
```

System Header

The format of the system header is:

```
typedef struct aouthdr
{
        short   magic;          /* magic number */
        short   vstamp;         /* version stamp */
        long    tsize;          /* text size in bytes, padded */
        long    dsize;          /* initialized data (.data) */
        long    bsize;          /* uninitialized data (.bss) */
        long    entry;          /* entry point */
        long    text_start;     /* base of text used for this file */
        long    data_start;     /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is:

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;  /* physical address */
        long            s_vaddr;  /* virtual address */
        long            s_size;   /* section size */
        long            s_scnptr; /* file ptr to raw data */
        long            s_relptr; /* file ptr to relocation */
        long            s_lnnoptr;/* file ptr to line numbers */
        unsigned short  s_nreloc; /* # reloc entries */
        unsigned short  s_nlnno;  /* # line number entries */
        long            s_flags;  /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long      r_vaddr;  /* (virtual) address of reference */
        long      r_symndx; /* index into symbol table */
        ushort    r_type;   /* relocation type */
};
```

The start of the relocation information is **s_relptr** from the section header. If there is no relocation information, **s_relptr** is 0.

Line Number

The **cc**(1V) command generates an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the **–g** option. Users can refer to line numbers when using the appropriate debugger, such as **dbx**(1)). The structure of these line number entries appears below.

```
struct lineno
{
        union
        {
                long    l_symndx ;
                long    l_paddr ;
        }               l_addr ;
        unsigned short  l_lnno ;
} ;
```

Numbering starts with one at the top of the source file and increments independent of transition between functions. The initial line number entry for a function has **l_lnno** equal to zero, and the symbol table index of the function's entry is in **l_symndx**. Otherwise, **l_lnno** is non-zero, and **l_paddr** is the physical address of the code for the referenced line. Thus the overall structure is the following:

l_addr	**l_lnno**
function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0

```
physical address        line
physical address        line
...
```

Symbol Table

The format of each symbol in the symbol table is described by the syment structure, shown below. This structure is compatible with System V COFF, but has an added **_n_dbx** structure which is needed by **dbx**(1).

```
#define SYMNMLEN 8
#define FILNMLEN 14
#define DIMNUM   4

struct syment
{
  union                              /* all ways to get a symbol name */
  {
    char           _n_name[SYMNMLEN]; /* name of symbol */
    struct
    {
      long         _n_zeroes;       /* == 0L if in string table */
      long         _n_offset;       /* location in string table */
    } _n_n;
    char           *_n_nptr[2];  /* allows overlaying */
    struct
    {
      char         _n_leading_zero; /* null char */
      char         _n_dbx_type;  /* stab type */
      short        _n_dbx_desc;  /* value of desc field */
      long         _n_stab_ptr;  /* table ptr */
    } _n_dbx;
  } _n;
  long             n_value;         /* value of symbol */
  short            n_scnum;         /* section number */
  unsigned short   n_type;          /* type and derived type */
  char             n_sclass;        /* storage class */
  char             n_numaux;        /* number of aux entries */
};

#define n_name     _n._n_name
#define n_zeroes   _n._n_n._n_zeroes
#define n_offset   _n._n_n._n_offset
#define n_nptr     _n._n_nptr[1]
```

The storage class member (**n_sclass**) is set to one of the constants defined in **<storclass.h>**. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows:

```
union auxent {
      struct {
            long     x_tagndx;
            union {
                  struct {
                        unsigned short  x_lnno;
                        unsigned short  x_size;
                  } x_lnsz;
                  long       x_fsize;
            } x_misc;
            union {
                  struct {
                        long     x_lnnoptr;
                        long     x_endndx;
                  } x_fcn;
                  struct {
                        unsigned short  x_dimen[DIMNUM];
                  } x_ary;
            } x_fcnary;
            unsigned short  x_tvndx;
      } x_sym;

      struct {
            char     x_fname[FILNMLEN];
      } x_file;

      struct {
            long          x_scnlen;
            unsigned short  x_nreloc;
            unsigned short  x_nlinno;
      } x_scn;

      struct {
            long                x_tvfill;
            unsigned short  x_tvlen;
            unsigned short  x_tvran[2];
      } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is **f_symptr** (from the file header) bytes from the beginning of the file. If the symbol table is stripped, **f_symptr** is 0. The string table (if one exists) begins at **f_symptr** + (**f_nsyms** * SYMESZ) bytes from the beginning of the file.

SEE ALSO

as(1), **cc**(1V), **ld**(1), **brk**(2), **ldfcn**(3)

NAME

core – format of memory image file

SYNOPSIS

#include <sys/core.h>

DESCRIPTION

The operating system writes out a memory image of a terminated process when any of various errors occur. See **sigvec**(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). Set-user-ID and set-group-ID programs do not produce core files when they terminate as this would cause a security loophole.

The maximum size of a **core** file is limited by **setrlimit** (see **getrlimit**(2)). Files which would be larger than the limit are not created.

The core file consists of a **core** structure, as defined in the **<sys/core.h>** file, followed by the data pages and then the stack pages of the process image. The **core** structure includes the program's header, the size of the text, data, and stack segments, the name of the program and the number of the signal that terminated the process. The program's header is described by the **exec** structure defined in the **<sys/exec.h>** file, except on Sun386i systems.

```
struct core {
        int     c_magic;          /* Corefile magic number */
        int     c_len;            /* Sizeof (struct core) */
        struct  regs c_regs;      /* General purpose registers */
        struct  exec c_aouthdr; /* A.out header */
        int     c_signo;          /* Killing signal, if any */
        int     c_tsize;          /* Text size (bytes) */
        int     c_dsize;          /* Data size (bytes) */
        int     c_ssize;          /* Stack size (bytes) */
        char    c_cmdname[CORE_NAMELEN + 1]; /* Command name */
        struct  fpu c_fpu;        /* external FPU state */
        int     c_ucode;          /* Exception no. from u_code */
};
```

The members of the structure are:

c_magic The magic number **CORE_MAGIC** , as defined in <**sys/core.h**>.

c_len The length of the **core** structure in the core file. This need not be equal to the current size of a **core** structure as defined in **<sys/core.h>**, as the core file may have been produced on a different release of the SunOS operating system.

c_regs The general purpose registers at the time the core file was produced. This structure is machine-dependent.

c_aouthdr The executable image header of the program.

c_signo The number of the signal that terminated the process; see **sigvec**(2).

c_tsize The size of the text segment of the process at the time the core file was produced.

c_dsize The size of the data segment of the process at the time the core file was produced. This gives the amount of data space image in the core file.

c_ssize The size of the stack segment of the process at the time the core file was produced. This gives the amount of stack space image in the core file.

c_cmdname The first **CORE_NAMELEN** characters of the last component of the path name of the program.

c_fpu The status of the floating point hardware at the time the core file was produced.

c_ucode The signal code of the signal that terminated the process, if any. See **sigvec**(2).

SEE ALSO

adb(1), **dbx**(1), **getrlimit**(2), **sigvec**(2)

NAME

cpio – format of cpio archive

DESCRIPTION

The old format *header* structure, when the **–c** option of **cpio** is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        ushort  h_ino,
                h_mode,
                h_uid,
                h_gid;
        short   h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to a word];
} Hdr;
```

The byte order here is that of the machine on which the tape was written. If the tape is being read on a machine with a different byte order, you have to use **swab**(3) after reading the header. You can determine what byte order the tape was written with by examining the *h_magic* field; if it is equal to 0143561 (octal), which is the standard magic number 070707 (octal) with the bytes swapped, the tape was written in a byte order opposite to that of the machine on which it is being read. If you are producing a tape to be read on a machine with the opposite byte order to that of the machine on which it is being produced, you can use **swap** before writing the header.

When the **–c** option is used, the *header* information is described by the statement below:

```
sscanf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Hdr.h_mtime, &Hdr.h_namesize, &Hdr.h_filesize, &Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file is recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in **stat**(2V). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name **TRAILER!!!**. Special files, directories, and the trailer, are recorded with *h_filesize* equal to zero. Symbolic links are recorded similarly to regular files, with the "contents" of the file being the name of the file the symbolic link points to.

SEE ALSO

cpio(1), **find**(1), **stat**(2V), **swab**(3)

NAME

crontab – table of times to run periodic jobs

SYNOPSIS

/var/spool/cron/crontabs/*

DESCRIPTION

The **cron** utility is a permanent process, started by **/etc/rc.local**. **cron** consults the files in the directory **/var/spool/cron/crontabs** to find out what tasks are to be done, and at what time.

Each line in a **crontab** file consists of six fields, separated by spaces or tabs, as follows:

minutes hours day-of-month month day-of-week command

minutes	Minutes field, which can have values in the range 0 through 59.
hours	Hours field, which can have values in the range 0 through 23.
day-of-month	Day of the month, in the range 1 through 31.
month	Month of the year, in the range 1 through 12.
day-of-week	Day of the week, in the range 0 through 6. Sunday is day 0 in this scheme of things. For backward compatibility with older systems, Sunday may also be specified as day 7.
command	Command to be run. A percent character in this field (unless escaped by \) is translated to a NEWLINE character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Any of fields 1 through 5 can be a list of values separated by commas. A value can either be a number, or a pair of numbers separated by a hyphen, indicating that the job is to be done for all the times in the specified range. If a field is an asterisk character (*) it means that the job is done for all possible values of the field.

Note: the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example,

0 0 1,15 * 1

would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to *. For example,

0 0 * * 1

would run a command only on Mondays.

The command is run from your home directory with an **arg0** of **sh**. Users who desire to have their **.profile** executed must explicitly do so in the command. **cron** supplies a default environment for every shell, defining **HOME, LOGNAME, USER, SHELL(=/bin/sh)**, and **PATH(=:/usr/ucb:/bin:/usr/bin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

Lines that start with # are treated as comments.

EXAMPLES

```
0 0 * * * calendar –
15 0 * * * /usr/etc/sa –s >/dev/null
15 4 * * * find /var/preserve –mtime +7 –a –exec rm –f {} ;
40 4 * * * find / –name '#*' –atime +3 –exec rm –f {} ;
0 0 * * 1-5 /usr/local/weekdays
0 0 * * 0,6 /usr/local/weekends
```

The **calendar** command runs at minute 0 of hour 0 (midnight) of every day. The **/usr/etc/sa** command runs at 15 minutes after midnight every day. The two **find** commands run at 15 minutes past four and at 40 minutes past four, respectively, every day of the year. The **/usr/local/weekdays** command is run at midnight on weekdays. Finally, the **/usr/local/weekends** command is run at midnight on weekends.

FILES

/var/spool/cron/crontabs/*
tables of times to run periodic jobs

/etc/rc.local

.profile

SEE ALSO

cron(8), **rc**(8)

NAME

dir – format of directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory and directories must be read using the **getdirentries**(2) system call or the **directory**(3V) library routines. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry; see **fs**(5).

A directory consists of some number of blocks of DIRBLKSIZ bytes, where DIRBLKSIZ is chosen such that it can be transferred to disk in a single atomic operation (512 bytes on most machines):

```
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255
```

Each DIRBLKSIZ byte block contains some number of directory entry structures, which are of variable length. Each directory entry has a **struct direct** at the front of it, containing its inode number, the length of the entry, and the length of the name contained in the entry. These are followed by the name padded to a 4-byte boundary with null bytes. All names are guaranteed null-terminated. The maximum length of a name in a directory is **MAXNAMLEN**.

The macro **DIRSIZ(dp)** gives the amount of space required to represent a directory entry. Free space in a directory is represented by entries that have:

```
dp->d_reclen > DIRSIZ(dp)
```

All DIRBLKSIZ bytes in a directory block are claimed by the directory entries. This usually results in the last entry in a directory having a large **dp->d_reclen**. When entries are deleted from a directory, the space is returned to the previous entry in the same directory block by increasing its **dp->d_reclen**. If the first entry of a directory block is free, then its **dp->d_ino** is set to 0. Entries other than the first in a directory do not normally have **dp->d_ino** set to 0.

The **DIRSIZ** macro gives the minimum record length which will hold the directory entry. This requires the amount of space in struct direct without the **d_name** field, plus enough space for the name with a terminating null byte **(dp->d_namlen+1)**, rounded up to a 4-byte boundary.

```
#undef DIRSIZ
#define DIRSIZ(dp)     ((sizeof (struct direct) - (MAXNAMLEN+1)) + (((dp)->d_namlen+1 + 3) &~ 3))
struct  direct {
        u_long  d_ino;
        short   d_reclen;
        short   d_namlen;
        char    d_name[MAXNAMLEN + 1];
        /* typically shorter */
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system ("/"), for which '..' has the same meaning as '.'.

SEE ALSO

getdirentries(2), **directory**(3V), **fs**(5)

NAME

dump, dumpdates – incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/inode.h>
#include <protocols/dumprestore.h>
```

DESCRIPTION

Tapes used by **dump** and **restore**(8) contain:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file **<protocols/dumprestore.h>** is:

```
#define TP_BSIZE            1024
#define NTREC               10
#define HIGHDENSITYTREC     32
#define CARTRIDGETREC       63
#define TP_NINDIR           (TP_BSIZE/2)

#define TS_TAPE             1
#define TS_INODE            2
#define TS_BITS             3
#define TS_ADDR             4
#define TS_END              5
#define TS_CLRI             6
#define OFS_MAGIC           (int)60011
#define NFS_MAGIC           (int)60012
#define CHECKSUM            (int)84446
union u_spcl {
        char dummy[TP_BSIZE];
        struct              s_spcl {
                            intc_type;
                            time_tc_date;
                            time_tc_ddate;
                            intc_volume;
                            daddr_tc_tapea;
                            ino_tc_inumber;
                            intc_magic;
                            intc_checksum;
                            structdinodec_dinode;
                            intc_count;
                            charc_addr[TP_NINDIR];
        } s_spcl;
} u_spcl;

#define spcl u_spcl.s_spcl

#define DUMPOUTFMT          "%-16s %c %s"/* for printf */
                            /* name, incno, ctime(date) */
#define DUMPINFMT           "%16s %c %[^\n]\n"/* inverse for scanf */
```

TP_BSIZE Size of file blocks on the dump tapes. Note: **TP_BSIZE** must be a multiple of **DEV_BSIZE.**

NTREC Default number of **TP_BSIZE** byte records in a physical tape block, changeable by the **b** option to **dump**.

HIGHDENSITYNTREC
Default number of **TP_BSIZE** byte records in a physical tape block on 6250 BPI or higher density tapes.

CARTRIDGETREC
Default number of **TP_BSIZE** records in a physical tape block on cartridge tapes.

TP_NINDIR Number of indirect pointers in a **TS_INODE** or **TS_ADDR** record. It must be a power of two.

The **TS_** entries are used in the **c_type** field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label

TS_INODE A file or directory follows. The **c_dinode** field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR A subrecord of a file description. See **c_addr** below.

TS_END End of tape record.

TS_CLRI A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.

NFS_MAGIC All header records have this number in **c_magic**.

CHECKSUM Header records checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the dump was taken.

c_ddate The date the file system was dumped from.

c_volume The current volume number of the dump.

c_tapea The current number of this (1024-byte) record.

c_inumber The number of the inode being dumped if this is of type **TS_INODE**.

c_magic This contains the value **MAGIC** above, truncated as needed.

c_checksum This contains whatever value is needed to make the record sum to **CHECKSUM**.

c_dinode This is a copy of the inode as it appears on the file system; see **fs**(5).

c_count The count of characters in **c_addr**.

c_addr An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, **TS_ADDR** records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a **TS_END** record and then the tapemark.

The dump history is kept in the file **/etc/dumpdates**. It is an ASCII file with three fields separated by white space:

The name of the device on which the dumped file system resides.

The level number of the dump tape; see **dump**(8).

The date of the incremental dump in the format generated by **ctime**(3V).

DUMPOUTFMT is the format to use when using **printf**(3S) to write an entry to **/etc/dumpdates**; **DUMPINFMT** is the format to use when using **scanf**(3S) to read an entry from **/etc/dumpdates**.

FILES

/etc/dumpdates

SEE ALSO

fs(5), **types**(5), **dump**(8), **restore**(8)

NAME

environ – user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

An array of strings called the 'environment' is made available by **execve**(2V) when a process begins. By convention these strings have the form '*name=value*'. The following names are used by various commands:

PATH The sequence of directory prefixes that **sh**(1), **time**(1V), **nice**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. The **login**(1) process sets **PATH=:/usr/ucb:/bin:/usr/bin.**

HOME The name of the user's login directory, set by **login**(1) from the password file **/etc/passwd** (see **passwd**(5)).

TERM The type of terminal on which the user is logged in. This information is used by commands, such as **nroff**(1) or **plot**(1G), which may exploit special terminal capabilities. See **/etc/termcap** (**termcap**(5)) for a list of terminal types.

SHELL The path name of the user's login shell.

TERMCAP The string describing the terminal in TERM, or the name of the **termcap** file, see **termcap**(3X), **termcap**(5).

EXINIT A startup list of commands read by **ex**(1), **edit**, and **vi**(1).

USER
LOGNAME The user's login name.

TZ The name of the time zone that the user is located in. If **TZ** is not present in the environment, the system's default time zone, normally the time zone that the computer is located in, is used.

Further names may be placed in the environment by the *export* command and '*name=value*' arguments in **sh**(1), or by the **setenv** command if you use **csh**(1). Arguments may also be placed in the environment at the point of an **execve**(2V). It is unwise to conflict with certain **sh**(1) variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS**.

SYSTEM V DESCRIPTION

The description of the variable **TERMCAP** does not apply to programs built in the System V environment.

FILES

/etc/passwd
etc/termcap

SEE ALSO

csh(1), **ex**(1), **login**(1), **nice**(1), **nroff**(1), **plot**(1G), **sh**(1), **time**(1V), **vi**(1), **execve**(2V), **getenv**(3V), **system**(3), **termcap**(3X), **passwd**(5), **termcap**(5)

NAME

ethers – Ethernet address to hostname database or NIS domain

DESCRIPTION

The **ethers** file contains information regarding the known (48 bit) Ethernet addresses of hosts on the Internet. For each host on an Ethernet, a single line should be present with the following information:

Ethernet-address official-host-name

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is "*x:x:x:x:x:x*" where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a SPACE, TAB, NEWLINE, or comment character. It is intended that host names in the **ethers** file correspond to the host names in the **hosts**(5) file.

The **ether_line**( ) routine from the Ethernet address manipulation library, **ethers**(3N) may be used to scan lines of the **ethers** file.

FILES

/etc/ethers

SEE ALSO

ethers(3N), **hosts**(5)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

exports, xtab – directories to export to NFS clients

SYNOPSIS

/etc/exports

/etc/xtab

DESCRIPTION

The **/etc/exports** file contains entries for directories that can be exported to NFS clients. This file is read automatically by the **exportfs**(8) command. If you change this file, you must run **exportfs**(8) for the changes to affect the daemon's operation.

Only when this file is present at boot time does the **rc.local** script execute **exportfs**(8) and start the NFS file-system daemon, **nfsd**(8).

The **/etc/xtab** file contains entries for directories that are *currently* exported. This file should only be accessed by programs using **getexportent** (see **exportent**(3)). (Use the **–u** option of **exportfs** to remove entries from this file).

An entry for a directory consists of a line of the following form:

directory –option[,*option*]...

directory is the pathname of a directory (or file).

option is one of

ro Export the directory read-only. If not specified, the directory is exported read-write.

rw=*hostnames*[**:***hostname*]**...**
Export the directory read-mostly. Read-mostly means read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all.

anon=*uid*
If a request comes from an unknown user, use *uid* as the effective user ID. Note: root users (uid 0) are always considered "unknown" by the NFS server, unless they are included in the "root" option below. The default value for this option is –2. Setting "anon" to –1 disables anonymous access. Note: by default secure NFS will accept insecure requests as anonymous, and those wishing for extra security can disable this feature by setting "anon" to –1.

root=*hostnames*[:*hostname*]**...**
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

access=*client*[:*client*]**...**
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see **netgroup**(5)). Each *client* in the list is first checked for in the netgroup database, and then the hosts database. The default value allows any machine to mount the given directory.

secure Require clients to use a more secure protocol when accessing the directory.

A '#' (pound-sign) anywhere in the file indicates a comment that extends to the end of the line.

EXAMPLE

```
/usr         -access=clients              # export to my clients
/usr/local   # export to the world
/usr2        -access=hermes:zip:tutorial  # export to only these machines
/usr/sun     -root=hermes:zip             # give root access only to these
/usr/new     -anon=0                      # give all machines root access
```

/usr/bin	**−ro**	**# export read-only to everyone**
/usr/stuff	**−access=zip,anon=−3,ro**	**# several options on one line**

FILES

/etc/exports
/etc/xtab
/etc/hosts
/etc/netgroup
rc.local

SEE ALSO

exportent(3), **hosts**(5), **netgroup**(5), **exportfs**(8), **nfsd**(8)

WARNINGS

You cannot export either a parent directory or a subdirectory of an exported directory that is *within the same filesystem*. It would be illegal, for instance, to export both **/usr** and **/usr/local** if both directories resided on the same disk partition.

NAME

ext_ports – external ports file for network printers, terminals, and modems

SYNOPSIS

/etc/ext_ports

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **ext_ports** external ports file is an ASCII file in the **/etc** directory on the Network Information Service (NIS) master server. **ext_ports** is used only by SNAP, and contains basic information about each printer, terminal, and modem on the network. This file contains a one-line entry for each device, and each field *must* be separated by a TAB character:

system:port type status baud model name #comment

system names the system to which the device is attached. This field contains only lower case and numeric characters, must start with a lower case character, and must not be longer than 32 characters.

port names the port in **/dev** on the *system*: **ttya** for the Sun386i serial port, **pp0** for the parallel port, and **ttym0** and **ttym1** for ports on an AT bus serial card.

type **printer**, **terminal**, or **modem**.

status indicates the device status. For terminals and printers, this can be **on** or **off**. An **off** status means the device is disabled from access by the SunOS operating system, but can still be accessed by DOS. For modems, this can be **in** to enable dialin, **out** to enable dialout, **in_out** to enable dialin and dialout, or **off**. An **off** status means the device is disabled from access by the SunOS operating system, but it can still be accessed by DOS.

baud is the baud rate.

model indicates the manufacturer or kind of device. For printers, this can be **epson**, **hp**, or **text**, for Epson and compatibles, HP Laserjet and compatibles, or for text-only printers. For terminals, this can be **vt100** or **wyse-50** for DEC VT-100 and compatibles or for Wyse WY-50 and compatibles. For modems, this can be **hayes** for Hayes and compatibles.

name is only used for unique naming of printers on the network. Up to 16 characters can be entered. This field is blank for terminals and modems — simply insert a TAB character.

#comment
can contain anything you want, up to a maximum of 96 characters.

EXAMPLE

In this example of an **ext_ports** file, the system vulcan has an epson printer attached to its parallel port, and a Wyse-50 terminal attached to its serial port, but with logins currently disabled. The system android has a VT100 attached to its serial port, with logins enabled. The system polaris has a **hayes** modem set for dialing out on an installed AT bus serial card.

vulcan:pp0	**printer**	**on**	**9600**	**epson**	**lp**	**#Engineering lab**
android:ttya	**terminal**	**on**	**9600**	**vt100**		**#Reception**
vulcan:ttya	**terminal**	**off**	**9600**	**wyse-50**		**#Engineering lab**
polaris:ttym0	**modem**	**in_out**	**2400**	**hayes**		**#QA lab**

FILES

/etc/ext_ports

SEE ALSO

snap(1), **vipw**(8)

Sun386i System and Network Administration,
Sun386i Advanced Administration

BUGS

The **/etc/ext_ports** file must be locked against simultaneous changes when it is edited; **vipw**(8) does the necessary locking.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

fbtab – framebuffer table

SYNOPSIS

/etc/fbtab

DESCRIPTION

The **/etc/fbtab** file contains information that is used by **login**(1), **getty**(8) and the window system (for example, **sunview**(1)) to change the owner, group, and permissions of window system devices upon logging into or out of a console device. By default, *all* lines in this file are commented out. That is, all window security is disabled. To enable window security, edit **/etc/fbtab**, log out, and log back in. You *must* edit this file before window security can be enabled.

The owner and group of the devices listed in **/etc/fbtab** are set to the owner and group of the console. The permissions are set as specified in **/etc/fbtab**. As in the example below, **0600** is the recommended permissions for normal security.

Fields are separated by TAB and/or SPACE characters. Blank lines and comments can appear anywhere in the file; comments are delimited by '#' and a NEWLINE.

The first field specifies the name of a console device (for example, **/dev/console**). The second field specifies the permissions to which the devices in the *device_list* field (third field) will be set. A *device_list* is a colon-separated list of device names (the full pathname is required).

Once the devices are owned by the user, their permissions and ownership can be changed using **chmod**(1V) and **chown**(8), as with any other user-owned file.

EXAMPLES

The following example entry in the **/etc/fbtab** file enables normal window security:

```
/dev/console 0600     /dev/kbd:/dev/mouse
/dev/console 0600     /dev/fb:/dev/bwone0:/dev/bwtwo0
/dev/console 0600     /dev/cgone0:/dev/cgtwo0:/dev/cgthree0:/dev/cgfour0
/dev/console 0600     /dev/cgsix0:/dev/cgeight0:/dev/cgnine0
/dev/console 0600     /dev/gpone0a:/dev/gpone0b:/dev/gpone0c:/dev/gpone0d
```

This entry specifies that upon login to **/dev/console**, the owner, group and permissions of all supported devices will be set to the user's username, the user's group and **0600**, respectively. You need only specify the devices supported by your configuration. Upon logout, the owner and group of these devices will be reset to **root** and **wheel**. The permissions remain as set in the **/etc/fbtab** file.

SEE ALSO

login(1), **sunview**(1), **sv_acquire**(1), **getty**(8)

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The **fcntl**(2V) function provides for control over open files. This include file describes *requests* and *arguments* to **fcntl** and **open**(2V) as shown below:

```
/*              @ (#)fcntl.h 1.2 83/12/08 SMI; from UCB 4.2 83/09/25*/
/*
* Flag values accessible to open(2V) and fcntl(2)
*  (The first three can only be set by open)
*/
#define         O_RDONLY              0
#define         O_WRONLY              1
#define         O_RDWR                2
#define         O_NDELAY              FNDELAY     /* Non-blocking I/O */
#define         O_APPEND              FAPPEND     /* append (writes guaranteed at the end) */
#ifndef         F_DUPFD
/* fcntl(2) requests */
#define         F_DUPFD               0           /* Duplicate fildes */
#define         F_GETFD               1           /* Get fildes flags */
#define         F_SETFD               2           /* Set fildes flags */
#define         F_GETFL               3           /* Get file flags */
#define         F_SETFL               4           /* Set file flags */
#define         F_GETOWN              5           /* Get owner */
#define         F_SETOWN              6           /* Set owner */
/* flags for F_GETFL, F_SETFL— copied from <sys/file.h> */
#define         FNDELAY                           00004/* non-blocking reads */
#define         FAPPEND                           00010/* append on each write */
#define         FASYNC                            00100/* signal pgrp when data ready */
#endif
```

SEE ALSO

fcntl(2V), **open**(2V)

NAME

fs, inode – format of a 4.2 (ufs) file system volume

SYNOPSIS

#include <sys/types.h>
#include <ufs/fs.h>
#include <ufs/inode.h>

DESCRIPTION

Standard 4.2 (ufs) file system storage volumes have a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the *super-block*. The layout of the super block is defined by the include file **<ufs/fs.h>**

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group contains inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of "blocks." File system blocks of at most size **MAXBSIZE** can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be **DEV_BSIZE**, or some multiple of a **DEV_BSIZE** unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the '**blksize(fs, ip, lbn)**' macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). The *lost+found* directory is given the next available inode when it is initially created by **mkfs**(8).

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of **fs_minfree** is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. **fs_nrpos** is the number of rotational positions which are distinguished. With the default **fs_nrpos** of 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for **fs_rotdelay** is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each **NBPI** bytes of disk space. The inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a **MINBSIZE** of 4096 it is possible to create files of size 2^32 with only two levels of indirection. **MINBSIZE** must be big enough to hold a cylinder group block, thus changes to **(struct cg)** must keep its size within **MINBSIZE.** Note: super blocks are never more than size **SBSIZE.**

The path name on which the file system is mounted is maintained in **fs_fsmnt. MAXMNTLEN** defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by **MAXCSBUFS.** It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from **fs_csaddr** (size **fs_cssize**) in addition to the super block.

Note: sizeof **(struct csum)** must be a power of two in order for the **fs_cs** macro to work.

inode: The inode is the focus of all file activity in the file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is "named" by its device/i-number pair. For further information, see the include file **<ufs/inode.h>**.

SEE ALSO

mkfs(8)

NAME

fspec – format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the operating system with non-standard tab stop settings, (that is, tab stops that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all TAB characters with the appropriate number of SPACE characters, before they can be processed by operating system commands. A format specification occurring in the first line of a text file specifies how TAB characters are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t *tabs* The **t** parameter specifies the tab stop settings for the file. The value of *tabs* must be one of the following:

- A list of column numbers separated by commas, indicating tab stops set at the specified columns;
- A '–' followed immediately by an integer *n*, indicating tab stops set at intervals of *n* columns, that is, at 1+*n*, 1+2**n*, and so on;
- A '–' followed by the name of a "canned" tab stop specification.

Up to 40 numbers are allowed in a comma-separated list of tab stop settings. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **t1, 10, 20, 30** and **t1, 10, +10, +10** are considered identical.

Standard tab stops are specified by **t–8**, or equivalently, **t1, 9, 17, 25,** etc. This is the tab stop setting that most operating system utilities assume, and is the most likely setting to be found at a terminal. The specification **t–0** specifies no tab stops at all.

The "canned" tab stops specifications that are recognized are as follows:

a 1, 10, 16, 36, 72
Assembler, IBM S/370, first format

a2 1, 10, 16, 40, 72
Assembler, IBM S/370, second format

c 1, 8, 12, 16, 20, 55
COBOL, normal format

c2 1, 6, 10, 14, 49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a TAB reaches column 12. Files using this tab stop setup should include a format specification as follows:
<:t–c2 m6 s66 d:>

c3 1, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 67
COBOL compact format (columns 1-6 omitted), with more tab stops than **c2**. This is the recommended format for COBOL. The appropriate format specification is:
<:t–c3 m6 s66 d:>

f 1, 7, 11, 15, 19, 23
FORTRAN

p 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61
PL/I

s 1, 10, 55
SNOBOL

u 1, 12, 20, 44
UNIVAC 1100 Assembler

s *size* The **s** parameter specifies a maximum line size. The value of **size** must be an integer. Size checking is performed after TAB characters have been expanded, but before the margin is prepended.

m *margin*
The **m** parameter specifies a number of SPACE characters to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

*** <:t5,10,15 s72:> ***

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1), **tabs**(1V)

NAME

fstab, mtab – static filesystem mounting table, mounted filesystems table

SYNOPSIS

/etc/fstab

/etc/mtab

DESCRIPTION

The **/etc/fstab** file contains entries for filesystems and disk partitions to mount using the **mount**(8) command, which is normally invoked by the **rc.boot** script at boot time. This file is used by various utilities that mount, unmount, check the consistency of, dump, and restore file systems. It is also used by the system itself when locating the swap partition.

The **/etc/mtab** file contains entries for filesystems *currently* mounted, and is read by programs using the routines described in **getmntent**(3). **umount** (see **mount**(8)) removes entries from this file.

Each entry consists of a line of the form:

filesystem directory type options freq pass

filesystem is the pathname of a block-special device, the name of a remote filesystem in *host*:*pathname* form, or the name of a "swap file" made with **mkfile**(8).

directory is the pathname of the directory on which to mount the filesystem.

type is the filesystem type, which can be one of:

- **4.2** to mount a block-special device
- **lo** to loopback-mount a file system
- **nfs** to mount an exported NFS filesystem
- **swap** to indicate a swap partition
- **ignore** to have the **mount** command ignore the current entry (good for noting disk partitions that are not being used)
- **rfs** to mount an RFS filesystem
- **tmp** filesystem in virtual memory
- **hsfs** to mount an ISO 9660 Standard or High Sierra Standard CD-ROM filesystem

options contains a comma-separated list (no spaces) of mounting options, some of which can be applied to all types of filesystems, and others which only apply to specific types.

4.2 options:

- **quota | noquota** Disk quotas are enforced or not enforced. The default is **noquota**.

nfs options:

- **bg | fg** If the first attempt fails, retry in the background, or, in the foreground.
- **noquota** Prevent **quota**(1) from checking whether the user is over quota on this file system; if the file system has quotas enabled on the server, quotas will still be checked for operations on this file system.
- **retry=***n* The number of times to retry the mount operation.
- **rsize=***n* Set the read buffer size to *n* bytes.
- **wsize=***n* Set the write buffer size to *n* bytes.
- **timeo=***n* Set the NFS timeout to *n* tenths of a second.
- **retrans=***n* The number of NFS retransmissions.
- **port=***n* The server IP port number.
- **soft | hard** Return an error if the server does not respond, or continue the retry request until the server responds.
- **intr** Allow keyboard interrupts on hard mounts.
- **secure** Use a more secure protocol for NFS transactions.

acregmin=*n*
Hold cached attributes for at least *n* seconds after file modification.

acregmax=*n*
Hold cached attributes for no more than *n* seconds after file modification.

acdirmin=*n*
Hold cached attributes for at least *n* seconds after directory update.

acdirmax=*n*
Hold cached attributes for no more than *n* seconds after directory update.

actimeo=*n*
Set *min* and *max* times for regular files and directories to *n* seconds.

noac Suppress attribute caching.

Regular defaults are:

fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,
acregmin=3,acregmax=60,acdirmin=30,acdirmax=60

actimeo has no default; it sets **acregmin**, **acregmax**, **acdirmin** and **acdirmax**

Defaults for **rsize** and **wsize** are set internally by the system kernel.

rfs options:

bg | fg If the first attempt fails, retry in the background, or, in the foreground.

retry=*n* The number of times to retry the mount operation.

Defaults are the same as for NFS.

Common options:

ro | rw mount either read-only or read-write

suid | nosuid
setuid execution allowed or disallowed

grpid Create files with BSD semantics for propagation of the group ID. With this option, files inherit the group ID of the directory in which they are created, regardless of the directory's setgid bit.

noauto Do not mount this file system automatically (using '**mount −a**').

freq is the interval (in days) between dumps.

pass is the **fsck**(8) pass in which to check the partition. Filesystems with *pass* 0 are not checked. Filesystems with the pass 1 are checked sequentially. In general, the root filesystem should be checked in pass 1, with others checked in higher (later) passes. For passes higher than 1, multiple filesystems in the same pass are checked simultaneously.

A hash-sign (#) as the first character indicates a comment line which is ignored by routines that read this file. The order of records in **/etc/fstab** is important because **fsck**, **mount**, and **umount** process the file sequentially; an entry for a file system must appear *after* the entry for any file system it is to be mounted on top of.

EXAMPLES

In this example, two partitions on the local disk are **4.2** mounted. Several **/export** directories are loopback mounted to appear in the traditional file system locations on the local system. The **/home/user** directory is hard mounted read-write over the NFS, along with additional swap space in the form of a mounted swap file (see *System and Network Administration* for details on adding swap space):

/dev/xy0a / 4.2 rw,noquota 1 1
/dev/xy0b /usr 4.2 rw,noquota 1 1
/export/tmp/localhost /tmp lo rw 0 0
/export/var/localhost /var lo rw 0 0
/export/cluster/sun386.sunos4.0.1 /usr/cluster lo rw 0 0
/export/local/sun386 /usr/local lo rw 0 0

example:/home/user /home/user nfs rw,hard,fg 0 0
/export/swap/myswap swap swap rw 0 0

FILES

/etc/fstab
/etc/mtab

SEE ALSO

swapon(2), **getmntent**(3), **lofs**(4S), **fsck**(8), **mkfile**(8), **mount**(8), **quotacheck**(8), **quotaon**(8), **swapon**(8)

System and Network Administration

NAME

ftpusers – list of users prohibited by FTP

SYNOPSIS

/etc/ftpusers

DESCRIPTION

ftpusers contains a list of users who cannot access this system using the File Transfer Protocol (FTP). **ftpusers** contains one user name per line.

If this file is missing, the list of users is considered to be empty, so that any user may use FTP to access the system if the other criteria for access are met (see **ftpd**(8C)).

SEE ALSO

ftp(1C), **ftpd**(8C)

NAME

gettytab – terminal configuration data base

SYNOPSIS

/etc/gettytab

DESCRIPTION

gettytab is a simplified version of the **termcap**(5) data base used to describe terminal lines. The initial terminal login process **getty**(8) accesses the **gettytab** file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, **default**, that is used to set global defaults for all other classes. That is, the **default** entry is read, then the entry for the class required is used to override particular settings.

CAPABILITIES

Refer to **termcap**(5) for a description of the file layout. The *Default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special default table.

Name	*Type*	*Default*	*Description*
ab	bool	false	read a \r first and guess the baud rate from it
ap	bool	false	terminal uses 7 bits, any parity
bd	num	0	backspace delay
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use crt backspace mode
cd	num	0	carriage-return delay
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add NEWLINE after login prompt
de	num	0	delay before first prompt is printed (seconds)
ds	str	^Y	delayed suspend character
dx	bool	false	set DECCTLQ
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses 7 bits, even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
ev	str	NULL	initial environment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
he	str	NULL	hostname editing string
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character
is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	^V	"literal next" character
lo	str	/usr/bin/login	program to exec when name obtained

ms	str	NULL	list of terminal modes to set or clear
m0	str	NULL	set modes that apply at the same time as those set by f0
m1	str	NULL	set modes that apply at the same time as those set by f1
m2	str	NULL	set modes that apply at the same time as those set by f2
nd	num	0	NEWLINE (LINEFEED) delay
nl	bool	false	terminal has (or might have) a NEWLINE character
nx	str	default	next table (for auto speed selection)
op	bool	false	terminal uses 7 bits, odd parity
os	num	unused	output speed
p8	bool	false	terminal uses 8 bits, no parity
pc	str		pad character
pe	bool	false	use printer (hard copy) erase algorithm
pf	num	0	delay between first prompt and following flush (seconds)
ps	bool	false	line connected to a MICOM port selector
qu	str	^	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use RAW for input, use CBREAK
sp	num	0	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
td	num	0	tab delay
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for environment)
ub	bool	false	do unbuffered output (of prompts etc)
uc	bool	false	terminal is known upper case only
we	str	^W	word erase character
xc	bool	false	do NOT echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when **getty** is entered. Specifying an input or output speed overrides line speed for stated direction only. If **ab** is specified, **getty** will initially read a character from the tty, assumed to be a carriage return, and will attempt to figure out the baud rate based on what the character appears as. It will then look for a table entry for that baud rate; if the line appears to be a 300 baud line, it will look for an entry **300-baud**, if it appears to be a 1200 baud line, it will look for an entry **1200-baud**, etc..

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the Boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overridden with one of the **f0**, **f1**, or **f2** numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

The **ms** field can be used to specify modes to be set and cleared. These modes are specified as **stty**(1V) modes; any mode supported by **stty** may be specified, except for the baud rate which must be specified with the **br** field. This permits modes not supported by the older terminal interface described in **ttcompat**(4M) to be set or cleared. Thus, to set the terminal port to which the printer is attached to even parity, TAB expansion, no NEWLINE to RETURN/LINEFEED translation, and RTS/CTS flow control enabled, do:

:ms=evenp,-tabs,nl,crtscts:

The **m0**, **m1**, and **m2** fields can be used to set modes which only apply concurrently with those set by **f0**, **f1**, and **f2**, respectively. The modes specified by **ms**, **m0**, **m1**, and **m2** are applied *after* the modes specified by other existing capabilities.

Should **getty** receive a null character (presumed to indicate a line break) it will restart using the table indicated by the **nx** entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The **cl** screen clear string may be preceded by a (decimal) number of milliseconds of delay required (as with **termcap**(5)). This delay is simulated by repeated use of the pad character **pc**.

The initial message, and login message, **im** and **lm** may include the character sequence **%h** or **%t** to obtain the hostname or tty name respectively. (**%%** obtains a single '**%**' character.) The hostname is normally obtained from the system, but may be set by the **hn** table entry. In either case it may be edited with **he**. The **he** string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the **he** string, copies one character from the real hostname to the final hostname. A '#' in the **he** string, skips the next character of the real hostname. Surplus '@' and '#' characters are ignored.

When **getty** execs the login process, given in the **lo** string (usually **/usr/bin/login**), it will have set the environment to include the terminal type, as indicated by the **tt** string (if it exists). The **ev** string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with **to**, then **getty** will exit within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from **getty** is even parity unless **op** or **p8** is specified. **op** may be specified with **ap** to allow any parity on input, but generate odd parity output. Note: this only applies while **getty** is being run, terminal driver limitations prevent a more complete implementation. **getty** does not check parity of input characters in RAW mode.

FILES

/etc/gettytab

SEE ALSO

termcap(5), **getty**(8)

NAME

group – group file

SYNOPSIS

/etc/group

DESCRIPTION

The **group** file contains a one-line entry for each group recognized by the system, of the form:

groupname:*password*:*gid*:*user-list*

where:

groupname	is the name of the group.
gid	is the group's numerical ID within the system; it must be unique.
user-list	is a comma-separated list of users allowed in the group.

If the password field is empty, no password is demanded. The **group** file is an ASCII file. Because of the encrypted passwords, the **group** file can and does have general read permission, and can be used as a mapping of numerical group IDs to group names.

A group entry beginning with a '+' (plus sign), means to incorporate an entry or entries from the Network Information Service (NIS) A '+' on a line by itself means to insert the entire contents of the NIS group file at that point in the file. An entry of the form: '+*groupname*' means to insert the entry (if any) for **groupname.** If a '+' entry has a non-empty *password* or *user-list* field, the contents of that field override the corresponding field from the NIS service. The *gid* field cannot be overridden in this way.

An entry of the form: –*groupname* indicates that the group is disallowed. All subsequent entries for the indicated *groupname*, whether originating from the NIS service, or the local **group** file, are ignored.

Malformed entries cause routines that read this file to halt, in which case group assignments specified further along are never made. To prevent this from happening, use **grpck**(8) to check the **/etc/group** database from time to time.

Sun386i systems uses the following group IDs as program privileges:

operator	5	Privilege to do backup as root.
accounts	11	Privilege to update user accounts.
networks	12	Privilege to change network configuration.
devices	13	Privilege to modify printer, terminal, or modem configurations.

On all Sun systems, SunOS uses group ID 0 as privilege to run **su**(1V).

EXAMPLE

Here is a sample group file when the **group.adjunct** file does not exist:

```
primary:q.mJzTnu8icF.:10:fred,mary
+myproject:::bill,steve
+:
```

Here is a sample group file when the **group.adjunct** file does exist:

```
primary:#$primary:10:fred,mary
+myproject:::bill,steve
+:
```

If these entries appear at the end of a group file, then the group *primary* will have members **fred** and **mary**, and a group ID of **10**. The group *myproject* will have members **bill** and **steve**, and the password and group ID of the NIS entry for the group **myproject**. All groups listed in the NIS service are pulled in and placed after the entry for **myproject**.

FILES

/etc/group

SEE ALSO

passwd(1), **su**(1V), **getgroups**(2V), **crypt**(3), **initgroups**(3), **group.adjunct**(5), **passwd**(5), **grpck**(8V)

NOTES

SunOS releases prior to SunOS 4.0, permitted a user to belong to no more then eight groups at a time. A user who belongs to more than eight groups may have trouble using the RPC service (and therefore NFS) to communicate with machines running older releases. In such cases, RPC complains of an "Authentication Error".

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

BUGS

The **passwd**(1) command will not change group passwords.

NAME

group.adjunct – group security data file

SYNOPSIS

/etc/security/group.adjunct

DESCRIPTION

The **group.adjunct** file contains the following information for each group:

groupname:*password*

groupname	The group's name in the system; it must be unique.
password	The encrypted password, formerly field two of the **/etc/group** file.

The **group.adjunct** file is in ASCII. Fields are separated by a colon, and each group is separated from the next by a NEWLINE.

A **group.adjunct** file can have a line beginning with a '+' (plus sign), which means to incorporate entries from the Network Information Service (NIS). There are two styles of '+' entries: all by itself, '+' means to insert the entire contents of the **group.adjunct** NIS file at that point; *+name* means to insert the entry (if any) for *name* from the NIS service at that point. If a '+' entry has a non-null password, the contents of that field will override what is contained in the NIS service.

FILES

/etc/group

SEE ALSO

crypt(3), **getgraent**(3), **getgrent**(3V), **group**(5)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME
help – help file format

SYNOPSIS
/usr/lib/help/*

AVAILABILITY
Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION
Each SunView application using the **help** feature has a simple ASCII file in **/usr/lib/help** with the name *application-name*.**info**.

This file contains the text of help messages for each SunView object within that program. Each help message is separated in the file by a line beginning with a colon and identified by a keyword which matches the HELP_DATA attribute of the SunView object.

The first character of each line in the file may be:

#	comment line
:	keyword line
any other	1-32 help text lines

If the line is a keyword line, it has the following structure—

:keyword[s]:datastring [pagenumber]NEWLINE

keyword is a 1-65 character keyword
- --any displayable characters may be used
- --several keywords may be present
- --keywords are separated by 1-or-more blanks

datastring is 1-256 ASCII bytes, and describes the path of the data files for help_viewer, relative to /usr/lib/help.

pagenumber is an optional page number within the help_viewer data file.

The help text which follows the :keyword line will be displayed in an Alert Box when help is requested for one of the keywords by pressing the help key.

The datastring will be sent (by RPC) to the **help_viewer** procedure when the user selects the More Help box in the Alert Box window.

EXAMPLE
Here is part of a typical help file, called **mailtool.info.**

```
:abort
  Abort button

o Quits the Mail application (click
  left on button). Tentative message
  deletions do not become permanent.

o Provides a menu of Abort options
  (click right on button).
```

:cancel:mailtool/Writing_and_Sending_Mail 1
Cancel button

o Closes the message composition
window without sending message
(click left on button).

o Provides a menu of Cancel options
(click right on button).

Pressing the help key while in the cancel or abort buttons triggers the display of the corresponding text. The words *cancel* and *abort* in this file are the keywords. In the case of abort, there is no More Help available. For cancel, More Help is available and it is stored in the first page of the **Writing_and_Sending_Mail** file in the mailtool directory.

FILES

/usr/lib/help/* files for the pop-up help facility

SEE ALSO

help_viewer(1), **help_viewer**(5)

Sun386i Developer's Guide

NAME

help_viewer – help viewer file format

SYNOPSIS

/usr/lib/help/*/*

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **help_viewer** reads files of various types. The Top Level list of applications documented is **/usr/lib/help/Top_Level**. The Master Index shown at the top level is **/usr/lib/help/Master_Index**. These files are FrameMaker files. To add or remove a heading from this list, use FrameMaker (1.1 or later).

Each directory within **/usr/lib/help** that corresponds to a SunView application name contains detailed information about that application. These are also FrameMaker files. The ***.rf** files are rasterfiles, of standard image format created by FrameMaker. These are the pictures that are interleaved into the text.

The **Frame/** subdirectory of **/usr/lib/help** contains topic, contents, and index templates which can be used to create new Help Viewer handbooks. The **Interleaf/** subdirectory contains Interleaf templates, fonts, and initialization files.

FILES

/usr/lib/help/*/*

SEE ALSO

help(5), **help_viewer**(1)

NAME

hosts – host name data base

SYNOPSIS

/etc/hosts

DESCRIPTION

The **hosts** file contains information regarding the known hosts on the TCP/IP. For each host a single line should be present with the following information:

Internet-address official-host-name aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional '.' notation using the **inet_addr ()** routine from the Internet address manipulation library, **inet**(3N). Host names may contain any printable character other than an upper case character, a field delimiter, NEWLINE, or comment character.

EXAMPLE

Here is a typical line from the **/etc/hosts** file:

192.9.1.20 gaia # John Smith

FILES

/etc/hosts

SEE ALSO

gethostent(3N), **inet**(3N)

NAME

hosts.equiv, .rhosts – trusted remote hosts and users

DESCRIPTION

The **/etc/hosts.equiv** and **.rhosts** files provide the "remote authentication" database for **rlogin**(1C), **rsh**(1C), **rcp**(1C), and **rcmd**(3N). The files specify remote hosts and users that are considered *trusted*. Trusted users are allowed to access the local system *without supplying a password.* The library routine **ruserok**( ) (see **rcmd**(3N)) performs the authentication procedure for programs by using the **/etc/hosts.equiv** and **.rhosts** files. The **/etc/hosts.equiv** file applies to the entire system, while individual users can maintain their own **.rhosts** files in their home directories.

These files *bypass* the standard password-based user authentication mechanism. To maintain system security, care must be taken in creating and maintaining these files.

The remote authentication procedure determines whether a particular remote user from a particular remote host should be allowed to access the local system as a (possibly different) particular local user. This procedure first checks the **/etc/hosts.equiv** file and then checks the **.rhosts** file in the home directory of the local user as whom access is being attempted. Entries in these files can be of two forms. *Positive* entries explicitly *allow* access, while *negative* entries explicitly *deny* access. The authentication succeeds as soon as a matching positive entry is found. The procedure fails when a matching negative entry is found, or if no matching entries are found in either file. The order of entries, therefore, can be important: If the files contain both matching positive and negative entries, the entry that appears first will prevail. The **rsh**(1C) and **rcp**(1C) programs fail if the remote authentication procedure fails. The **rlogin** program will fall back to the standard password-based login procedure if the remote authentication fails.

Both files are formatted as a list of one-line entries. Each entry has the form:

> *hostname* [*username*]

Negative entries are differentiated from positive entries by a '-' character preceding either the *hostname* or *username* field.

Positive Entries

If the form:

> *hostname*

is used, then users from the named host are trusted. That is, they may access the system with the same user name as they have on the remote system. This form may be used in both the **/etc/hosts.equiv** and **.rhosts** files.

If the line is in the form:

> *hostname username*

then the named user from the named host can access the system. This form may be used in individual **.rhosts** files to allow remote users to access the system *as a different local user.* If this form is used in the **/etc/hosts.equiv** file, the named remote user will be allowed to access the system as *any* local user.

Netgroups(5) can be used in either the *hostname* or *username* fields to match a number of hosts or users in one entry. The form:

> *+@netgroup*

allows access from all hosts in the named netgroup. When used in the *username* field, netgroups allow a group of remote users to access the system as a particular local user. The form:

> *hostname +@netgroup*

allows all of the users in the named netgroup from the named host to access the system as the local user. The form:

> *+@netgroup1 +@netgroup2*

allows the users in *netgroup2* from the hosts in *netgroup1* to access the system as the local user.

The special character '+' can be used in place of either *hostname* or *username* to match any host or user. For example, the entry

+

will allow a user from any remote host to access the system with the same username. The entry

+ *username*

will allow the named user from any remote host to access the system. The entry

hostname +

will allow any user from the named host to access the system as the local user.

Negative Entries

Negative entries are preceded by a '-' sign. The form:

-hostname

will disallow all access from the named host. The form:

−@netgroup

means that access is explicitly disallowed from all hosts in the named netgroup. The form:

hostname −username

disallows access by the named user only from the named host, while the form:

+ *−@netgroup*

will disallow access by all of the users in the named netgroup from all hosts.

FILES

/etc/hosts.equiv
~/.rhosts

NOTES

Hostnames in **/etc/hosts.equiv** and **.rhosts** files must be the "official" name of the host, not one of its nicnames.

Root access is handled as a special case. Only the */.rhosts* file is checked when the access is being attempted for root. To help maintain system security, the */etc/hosts.equiv* file is not checked.

As a security feature, the *.rhosts* file must be owned by the user as whom access is being attempted.

Positive entries in */etc/hosts.equiv* that include a *username* field (either an individual named user, a netgroup, or '+' sign) should be used only with extreme caution. Because */etc/hosts.equiv* applies system-wide, these entries allow one or a group of remote users to access the system *as any local user*. This can be the source of a security hole.

SEE ALSO

rlogin(1C), **rsh**(1C), **rcp**(1C), **rcmd**(3N), **hosts**(5), **netgroup**(5), **passwd**(5)

NAME

indent.pro – default options for indent

DESCRIPTION

The **.indent.pro** file in either the current or home directory contains default command line options for the **indent**(1) program. It is a text file that contains space-separated command line options. For a description of these options, see **indent**(1).

Explicit command line options override options taken from **.indent.pro**.

Here is a sample **.indent.pro** file:

–bap –nbad –nbbb –bc –br –cdb –nce
–fc1 –ip –lp –npcs –psl –sc –nsob –cli0
–di12 –l79 –i4 –d0 –c33

FILES

./.indent.pro
~/.indent.pro

SEE ALSO

indent(1)

NAME

inetd.conf – Internet servers database

DESCRIPTION

The **inetd.conf** file contains the list of servers that **inetd**(8C) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

service-name socket-type protocol wait-status uid server-program server-arguments

Fields can be separated by either spaces or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

service-name is the name of a valid service listed in the file **/etc/services**. For RPC services, the value of the *service-name* field consists of the RPC service name, followed by a slash and either a version number or a range of version numbers (for example, **mountd/1**).

socket-type can be one of:

stream	for a stream socket,
dgram	for a datagram socket,
raw	for a raw socket,
rdm	for a "reliably delivered message" socket, or
seqpacket	for a sequenced packet socket.

protocol must be a recognized protocol listed in the file **/etc/protocols**. For RPC services, the field consists of the string "rpc" followed by a slash and the name of the protocol (for example, **rpc/udp** for an RPC service using the UDP protocol as a transport mechanism).

wait-status is **nowait** for all but "single-threaded" datagram servers — servers which do not release the socket until a timeout occurs (such as **comsat**(8C) and **talkd**(8C)). These must have the status **wait**. Although **tftpd**(8C) establishes separate "pseudo-connections", its forking behavior can lead to a race condition unless it is also given the status **wait**.

uid is the user ID under which the server should run. This allows servers to run with access privileges other than those for root.

server-program is either the pathname of a server program to be invoked by **inetd** to perform the requested service, or the value **internal** if **inetd** itself provides the service.

server-arguments If a server must be invoked with command-line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects **inetd** to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '**%A**'.

FILES

/etc/inetd.conf
/etc/services
/etc/protocols

SEE ALSO

services(5), **comsat**(8C), **inetd**(8C), **talkd**(8C), **tftpd**(8C)

BUGS

inetd dumps core when the **inetd.conf** file contains blank lines.

NAME

internat – key mapping table for internationalization

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

This file format is used for the file specified by the –f option of **old-setkeys**(1).

The file has three columns. First column is keytable identifier, one of: BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG. The second column is a decimal keystation number. The third column is hexadecimal keytable entry value. The file must end with line of "END, 0, 0". As usual, comment lines start with #.

EXAMPLES

This is the file for mapping keys to Canadian standards:

```
# /usr/lib/.setkeys: Key remapping, used by "setkeys remap"
#
# First column is keytable identifier:
#               BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG
# Second column is decimal keystation number
# Third column is hexadecimal keytable entry value
# File must end with line of "END, 0, 0"
# Comment lines must start with #
#
#
# --- Keymaps for Canadian keyboard ---
# > Define Alt Graph key (SHIFTKEYS+ALTGRAPH=86)
BASE 119 86
CTRL 119 86
SHIFT 119 86
CAPS 119 86
UP 119 86
# > Define Caps key (SHIFTKEYS+CAPSLOCK=80)
BASE 13 80
CTRL 13 80
SHIFT 13 80
CAPS 13 80
# > Define Floating Accent keys
#               FA_UMLAUT = A9
#               FA_CFLEX = AA
#               FA_TILDE = AB
#               FA_CEDILLA = AC
#               FA_ACUTE = AD
#               FA_GRAVE = AE
BASE 64 AA
SHIFT 64 A9
CAPS 64 A9
BASE 65 AC
SHIFT 65 AB
CAPS 65 AB
BASE 87 AE
SHIFT 87 AD
CAPS 87 AD
# > Define ASCII values
```

```
BASE 88 5B
SHIFT 88 7B
CAPS 88 7B
BASE 15 5D
SHIFT 15 7D
CAPS 15 7D
SHIFT 31 22
SHIFT 32 2F
SHIFT 35 3F
SHIFT 107 27
CAPS 107 27
SHIFT 108 60
CAPS 108 60
BASE 124 3C
SHIFT 124 3E
CAPS 124 3E
# > Define ISO values
BASE_ISO 109 E9
SHIFT_ISO 109 C9
# > Define Alternate Graph ISO values
ALTG 88 AB
ALTG 15 BB
ALTG 30 B1
ALTG 31 B2
ALTG 32 B3
ALTG 33 A2
ALTG 34 A4
ALTG 35 5E
ALTG 36 40
ALTG 37 A3
ALTG 38 5C
ALTG 40 AC
ALTG 41 23
ALTG 63 B6
ALTG 64 BC
ALTG 65 BD
ALTG 42 BE
ALTG 106 B5
ALTG 105 BA
# > End of file
END 0 0
```

SEE ALSO

old-setkeys(1)

The *Sun386i Developer's Guide* for keystation number diagrams.

NAME

ipalloc.netrange – range of addresses to allocate

SYNOPSIS

/etc/ipalloc.netrange

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

This file, if it exists on the Network Information Service (NIS) master of the **hosts.byaddr** map, specifies the ranges of IP addresses that can be allocated by the **ipallocd**(8C) daemon. This allows multiple address assignment authorities, probably in multiple administrative domains, to coexist on the same IP network by preallocating ranges of addresses. If this file does not exist, the daemon assumes that all addresses not listed in the **hosts** map may be freely allocated.

This file can contain blank lines. Comments begin with a '#' character and extend to the end of the current line. Ranges of free addresses are specified on one line per network or subnetwork.

The first token on the line is the IP address, in four part "dot" notation as also used in the **hosts** file, of the network or subnetwork described. It is separated from the second token by white space. The second token is a comma-separated list of local host number ranges on that network. These ranges take two forms: a single number specifies just that local host number, and two numbers separated by a dash specify all local host numbers starting at the first number and ending at the second. In the case of a subnet, host numbers not in that subnet are excluded.

For example, the following file would specify that a subset of the addresses on the class C network 192.9.200.0 may be allocated, and only some of the addresses on two particular subnets of the class B network 128.255.0.0 may be allocated. In any case, only non-broadcast addresses not listed in the **hosts** map are subject to allocation:

```
# We have three network cables administered using automatic # IP address allocation.

192.9.200.0         50-100,200-254        128.255.210.0        3,5,7,9,100-110
128.255.211.0       1-254
```

SEE ALSO

hosts(5), **netmasks**(5), **ipallocd**(8C)

BUGS

There is a silent limit of twenty ranges per network.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

keytables – keyboard table descriptions for loadkeys and dumpkeys

DESCRIPTION

These files are used by **loadkeys**(1) to modify the translation tables used by the keyboard streams module **kb**(4M), and generated by **dumpkeys** (see **loadkeys**(1)) from those translation tables.

Any line in the file beginning with # is a comment, and is ignored. # is treated specially only at the beginning of a line.

Other lines specify the values to load into the tables for a particular keystation. The format is either:

key *number list_of_entries*

or

swap *number1* **with** *number2*

or

key *number1* **same as** *number2*

or a blank line, which is ignored.

key *number list_of_entries*

sets the entries for keystation *number* from the list given. An entry in that list is of the form

tablename code

where *tablename* is the name of a particular translation table, or **all**. The translation tables are:

base	entry when no shifts are active
shift	entry when "Shift" key is down
caps	entry when "Caps Lock" is in effect
ctrl	entry when "Control" is down
altg	entry when "Alt Graph" is down
numl	entry when "Num Lock" is in effect
up	entry when a key goes up

All tables other than **up** refer to the action generated when a key goes down. Entries in the **up** table are used only for shift keys, since the shift in question goes away when the key goes up, except for keys such as "Caps Lock" or "Num Lock"; the keyboard streams module makes the key look as if it were a latching key.

A table name of **all** indicates that the entry for all tables should be set to the specified value, with the following exception: for entries with a value other than **hole**, the entry for the **numl** table should be set to **nonl**, and the entry for the **up** table should be set to **nop**.

The *code* specifies the effect of the key in question when the specified shift key is down. A *code* consists of either:

- A character, which indicates that the key should generate the given character. The character can either be a single character, a single character preceded by ^ which refers to a "control character" (for instance, **^c** is control-C), or a C-style character constant enclosed in single quote characters ('), which can be expressed with C-style escape sequences such as \r for RETURN or \000 for the null character. Note that the single character may be any character in an 8-bit character set, such as ISO 8859/1.
- A string, consisting of a list of characters enclosed in double quote characters ("). Note that the use of the double quote character means that a *code* of double quote must be enclosed in single quotes.

- One of the following expressions:

shiftkeys+leftshift
the key is to be the left-hand "Shift" key

shiftkeys+rightshift
the key is to be the right-hand "Shift" key

shiftkeys+leftctrl
the key is to be the left-hand "Control" key

shiftkeys+rightctrl
the key is to be the right-hand "Control" key

shiftkeys+alt the key is to be the "Alt" shift key

shiftkeys+altgraph
the key is to be the "Alt Graph" shift key

shiftkeys+capslock
the key is to be the "Caps Lock" key

shiftkeys+shiftlock
the key is to be the "Shift Lock" key

shiftkeys+numlock
the key is to be the "Num Lock" key

buckybits+systembit
the key is to be the "Stop" key in Sunview; this is normally the L1 key, or the SETUP key on the VT100 keyboard

buckybits+metabit
the key is to be the "meta" key, that is, the "Left" or "Right" key on a Sun-2 or Sun-3 keyboard or the "diamond" key on a Sun-4 keyboard

compose the key is to be the "Compose" key

ctrlq on the "VT100" keyboard, the key is to transmit the control-Q character (this would be the entry for the "Q" key in the **ctrl** table)

ctrls on the "VT100" keyboard, the key is to transmit the control-S character (this would be the entry for the "S" key in the **ctrl** table)

noscroll on the "VT100" keyboard, the key is to be the "No Scroll" key

string+uparrow the key is to be the "up arrow" key

string+downarrow
the key is to be the "down arrow" key

string+leftarrow the key is to be the "left arrow" key

string+rightarrow
the key is to be the "right arrow" key

string+homearrow
the key is to be the "home" key

fa_acute the key is to be the acute accent "floating accent" key

fa_cedilla the key is to be the cedilla "floating accent" key

fa_cflex the key is to be the circumflex "floating accent" key

fa_grave the key is to be the grave accent "floating accent" key

fa_tilde the key is to be the tilde "floating accent" key

fa_umlaut	the key is to be the umlaut "floating accent" key
nonl	this is used only in the Num Lock table; the key is not to be affected by the state of Num Lock
pad0	the key is to be the "0" key on the numeric keypad
pad1	the key is to be the "1" key on the numeric keypad
pad2	the key is to be the "2" key on the numeric keypad
pad3	the key is to be the "3" key on the numeric keypad
pad4	the key is to be the "4" key on the numeric keypad
pad5	the key is to be the "5" key on the numeric keypad
pad6	the key is to be the "6" key on the numeric keypad
pad7	the key is to be the "7" key on the numeric keypad
pad8	the key is to be the "8" key on the numeric keypad
pad9	the key is to be the "9" key on the numeric keypad
paddot	the key is to be the "." key on the numeric keypad
padenter	the key is to be the "Enter" key on the numeric keypad
padplus	the key is to be the "+" key on the numeric keypad
padminus	the key is to be the "-" key on the numeric keypad
padstar	the key is to be the "*" key on the numeric keypad
padslash	the key is to be the "/" key on the numeric keypad
padequal	the key is to be the "=" key on the numeric keypad
padsep	the key is to be the "," (separator) key on the numeric keypad
lf(*n*)	the key is to be the left-hand function key *n*
rf(*n*)	the key is to be the right-hand function key *n*
tf(*n*)	the key is to be the top function key *n*
bf(*n*)	the key is to be the "bottom" function key *n*
nop	the key is to do nothing
error	this code indicates an internal error; to be used only for keystation 126, and must be used there
idle	this code indicates that the keyboard is idle (that is, has no keys down); to be used only for all entries other than the **numl** and **up** table entries for keystation 127, and must be used there
oops	this key exists, but its action is not defined; it has the same effect as **nop**
reset	this code indicates that the keyboard has just been reset; to be used only for the **up** table entry for keystation 127, and must be used there

swap *number1* **with** *number2*

exchanges the entries for keystations *number1* and *number2*.

key *number1* **same as** *number2*

sets the entries for keystation *number1* to be the same as those for keystation *number2*. If the file does not specify entries for keystation *number2*, the entries currently in the translation table are used; if the file does specify entries for keystation *number2*, those entries are used.

EXAMPLES

The following entry sets keystation 15 to be a "hole" (that is, an entry indicating that there is no keystation 15); sets keystation 30 to do nothing when Alt Graph is down, generate "!" when Shift is down, and generate "1" under all other circumstances; and sets keystation 76 to be the left-hand Control key.

```
key 15	all hole
key 30	base 1 shift ! caps 1 ctrl 1 altg nop
key 76	all shiftkeys+leftctrl up shiftkeys+leftctrl
```

The following entry exchanges the Delete and Back Space keys on the Type 4 keyboard:

```
swap 43 with 66
```

Keystation 43 is normally the Back Space key, and keystation 66 is normally the Delete key.

The following entry disables the Caps Lock key on the Type 3 and U.S. Type 4 keyboards:

```
key 119 all nop
```

The following specifies the standard translation tables for the U.S. Type 4 keyboard:

```
key 0	all hole
key 1	all buckybits+systembit up buckybits+systembit
key 2	all hole
key 3	all lf(2)
key 4	all hole
key 5	all tf(1)
key 6	all tf(2)
key 7	all tf(10)
key 8	all tf(3)
key 9	all tf(11)
key 10	all tf(4)
key 11	all tf(12)
key 12	all tf(5)
key 13	all shiftkeys+altgraph up shiftkeys+altgraph
key 14	all tf(6)
key 15	all hole
key 16	all tf(7)
key 17	all tf(8)
key 18	all tf(9)
key 19	all shiftkeys+alt up shiftkeys+alt
key 20	all hole
key 21	all rf(1)
key 22	all rf(2)
key 23	all rf(3)
key 24	all hole
key 25	all lf(3)
key 26	all lf(4)
key 27	all hole
key 28	all hole
key 29	all ^[
key 30	base 1 shift ! caps 1 ctrl 1 altg nop
key 31	base 2 shift @ caps 2 ctrl ^@ altg nop
key 32	base 3 shift # caps 3 ctrl 3 altg nop
key 33	base 4 shift $ caps 4 ctrl 4 altg nop
key 34	base 5 shift % caps 5 ctrl 5 altg nop
key 35	base 6 shift ^ caps 6 ctrl ^^ altg nop
key 36	base 7 shift & caps 7 ctrl 7 altg nop
```

```
key 37   base 8 shift * caps 8 ctrl 8 altg nop
key 38   base 9 shift ( caps 9 ctrl 9 altg nop
key 39   base 0 shift ) caps 0 ctrl 0 altg nop
key 40   base - shift _ caps - ctrl ^_ altg nop
key 41   base = shift + caps = ctrl = altg nop
key 42   base ` shift ~ caps ` ctrl ^^ altg nop
key 43   all '\b'
key 44   all hole
key 45   all rf(4) numl padequal
key 46   all rf(5) numl padslash
key 47   all rf(6) numl padstar
key 48   all bf(13)
key 49   all lf(5)
key 50   all bf(10) numl padequal
key 51   all lf(6)
key 52   all hole
key 53   all '\t'
key 54   base q shift Q caps Q ctrl ^Q altg nop
key 55   base w shift W caps W ctrl ^W altg nop
key 56   base e shift E caps E ctrl ^E altg nop
key 57   base r shift R caps R ctrl ^R altg nop
key 58   base t shift T caps T ctrl ^T altg nop
key 59   base y shift Y caps Y ctrl ^Y altg nop
key 60   base u shift U caps U ctrl ^U altg nop
key 61   base i shift I caps I ctrl '\t' altg nop
key 62   base o shift O caps O ctrl ^O altg nop
key 63   base p shift P caps P ctrl ^P altg nop
key 64   base [ shift { caps [ ctrl ^[ altg nop
key 65   base ] shift } caps ] ctrl ^] altg nop
key 66   all '\177'
key 67   all compose
key 68   all rf(7) numl pad7
key 69   all rf(8) numl pad8
key 70   all rf(9) numl pad9
key 71   all bf(15) numl padminus
key 72   all lf(7)
key 73   all lf(8)
key 74   all hole
key 75   all hole
key 76   all shiftkeys+leftctrl up shiftkeys+leftctrl
key 77   base a shift A caps A ctrl ^A altg nop
key 78   base s shift S caps S ctrl ^S altg nop
key 79   base d shift D caps D ctrl ^D altg nop
key 80   base f shift F caps F ctrl ^F altg nop
key 81   base g shift G caps G ctrl ^G altg nop
key 82   base h shift H caps H ctrl '\b' altg nop
key 83   base j shift J caps J ctrl '\n' altg nop
key 84   base k shift K caps K ctrl '\v' altg nop
key 85   base l shift L caps L ctrl ^L altg nop
key 86   base ; shift : caps ; ctrl ; altg nop
key 87   base '\'' shift '"' caps '\'' ctrl '\'' altg nop
key 88   base '\\' shift | caps '\\' ctrl ^\ altg nop
key 89   all '\r'
```

```
key 90   all bf(11) numl padenter
key 91   all rf(10) numl pad4
key 92   all rf(11) numl pad5
key 93   all rf(12) numl pad6
key 94   all bf(8) numl pad0
key 95   all lf(9)
key 96   all hole
key 97   all lf(10)
key 98   all shiftkeys+numlock
key 99   all shiftkeys+leftshift up shiftkeys+leftshift
key 100  base z shift Z caps Z ctrl ^Z altg nop
key 101  base x shift X caps X ctrl ^X altg nop
key 102  base c shift C caps C ctrl ^C altg nop
key 103  base v shift V caps V ctrl ^V altg nop
key 104  base b shift B caps B ctrl ^B altg nop
key 105  base n shift N caps N ctrl ^N altg nop
key 106  base m shift M caps M ctrl '\r' altg nop
key 107  base , shift < caps , ctrl , altg nop
key 108  base . shift > caps . ctrl . altg nop
key 109  base / shift ? caps / ctrl ^_ altg nop
key 110  all shiftkeys+rightshift up shiftkeys+rightshift
key 111  all '\n'
key 112  all rf(13) numl pad1
key 113  all rf(14) numl pad2
key 114  all rf(15) numl pad3
key 115  all hole
key 116  all hole
key 117  all hole
key 118  all lf(16)
key 119  all shiftkeys+capslock
key 120  all buckybits+metabit up buckybits+metabit
key 121  base ' ' shift ' ' caps ' ' ctrl ^@ altg ' '
key 122  all buckybits+metabit up buckybits+metabit
key 123  all hole
key 124  all hole
key 125  all bf(14) numl padplus
key 126  all error numl error up hole
key 127  all idle numl idle up reset
```

SEE ALSO

loadkeys(1), **kb**(4M)

NAME

link – link editor interfaces

SYNOPSIS

#include <link.h>

DESCRIPTION

Dynamically linked executables created by **ld**(1) contain data structures used by the dynamic link editor to finish link-editing the program during program execution. These data structures are described with a **link_dynamic** structure, as defined in the **link.h** file. **ld** always identifies the location of this structure in the executable file with the symbol **__DYNAMIC**. This symbol is **ld**-defined and if referenced in an executable that does not require dynamic linking will have the value zero.

The program stub linked with "main" programs by compiler drivers such as **cc**(1V) (called **crt0**) tests the definition of **__DYNAMIC** to determine whether or not the dynamic link editor should be invoked. Programs supplying a substitute for **crt0** must either duplicate this functionality or else require that the programs with which they are linked be linked *statically*. Otherwise, such replacement **crt0**'s must open and map in the executable **/usr/lib/ld.so** using **mmap**(2). Care should be taken to ensure that the expected mapping relationship between the "text" and "data" segments of the executable is maintained in the same manner that the **execve**(2V) system call does. The first location following the **a.out** header of this executable is the entry point to a function that begins the dynamic link-editing process. This function must be called and supplied with two arguments. The first argument is an integer representing the revision level of the argument list, and should have the value "1". The second should be a pointer to an argument list structure of the form:

```
struct {
        int      crt_ba;                  /* base address of ld.so */
        int      crt_dzfd;                /* open fd to /dev/zero */
        int      crt_ldfd;                /* open fd to ld.so */
        struct   link_dynamic *crt_dp;    /* pointer to program's __DYNAMIC */
        char     **crt_ep;                /* environment strings */
        caddr_t  crt_bp;                  /* debugger hook */
}
```

The members of the structure are:

crt_ba The address at which **/usr/lib/ld.so** has been mapped.

crt_dzfd An open file descriptor for **/dev/zero**. **ld.so** will close this file descriptor before returning.

crt_ldfd The file descriptor used to map **/usr/lib/ld.so**. **ld.so** will close this file descriptor before returning.

crt_dp A pointer to the label **__DYNAMIC** in the executable which is calling **ld.so**.

crt_ep A pointer to the environment strings provided to the program.

crt_bp A location in the executable which contains an instruction that will be executed after the call to **ld.so** returns. This location is used as a breakpoint in programs that are being executed under the control of a debugger such as **adb**(1).

SEE ALSO

ld(1), **mmap**(2), **a.out**(5)

BUGS

These interfaces are under development and are subject to rapid change.

NAME

locale – locale database

SYNOPSIS

/usr/share/lib/locale/*category/locale*

/etc/locale/*category/locale*

DESCRIPTION

The *category* directory contains information relating to one category of the complete list of categories that comprise a full locale for all systems sharing this directory. *locale* is either a file or a directory that contains information relating to the relevant category indicated by its parent directory *category*. *locale* is the name that is given to describe the style of operation required by an application in a particular language, territory or code-set.

At runtime these directories will be accessed if the application has made a valid call to:

setlocale(*category, locale***)**

where *category* can be any one of the following settings:

LC_COLLATE Collation order. Affects the behavior of regular expressions and the string functions defined in **strcoll**(3).

LC_CTYPE Character classification and case conversion. Affects the behavior of regular expressions and the character handling functions defined in **toascii**(3), and **ctime**(3V).

LC_MONETARY Monetary formatting. Affects the behavior of functions that handle monetary values.

LC_NUMERIC Numeric delimiters. Affects the radix character of the formatted input/output functions defined in **printf**(3V) and **scanf**(3V), and the conversion functions defined in **strtod**(3).

LC_TIME Date and time formats. Affects the behavior of the time functions defined in **ctime**(3V).

LC_MESSAGES Message presentation style. Affects the behavior of the string access functions defined in **catgets**(3C) and **gettext**(3).

NLSPATH Contains a sequence of pseudo-pathnames which **catopen**(3C) uses when attempting to locate message catalogs. Each pseudo-pathname contains a name template consisting of an optional path-prefix, one or more substitution fields, a filename and an optional filename suffix.

Substitution fields consist of a **%** symbol, followed by a single-letter keyword. The following keywords are currently defined:

%N The value of the *name* parameter passed to **catopen**(3C).

%L The value of the **LANG** environment variable.

%% A single **%** character.

A null string is sustituted if the specified value is not defined. Pathnames defined in **NLSPATH** are separated by colons (**:**). A leading or two adjacent colons indicate the current directory. For example:

NLSPATH=":%N.cat:/nlslib/%L/%N.cat"

Indicates to **catopen**(3C) that it should look for the requested message catalog in *name, name.cat* and */nlslib/***$LANG***/name.cat.* The **LC_ALL** and **LANG** environment variables do not commute to real directories or files but instead relate to a locale that is a assumed to be valid for all of the above categories.

SEE ALSO

catgets(3C), **catopen**(3C), **ctime**(3V), **gettext**(3), **printf**(3V), **scanf**(3V), **setlocale**(3V), **strcoll**(3) **strtod**(3), **toascii**(3V)

NAME

magic – file command's magic number file

DESCRIPTION

The **file**(1) command identifies the type of a file using, among other tests, a test for whether the file begins with a certain *magic number*. The file **/etc/magic** specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

offset *type* *value* *message*

offset A number specifying the offset, in bytes, into the file of the data which is to be tested.

type The type of the data to be tested. The possible values are:

- **byte** A one-byte value.
- **short** A two-byte value.
- **long** A four-byte value.
- **string** A string of bytes.

The types **byte**, **short**, and **long** may optionally be followed by a mask specifier of the form &*number*. If a mask specifier is given, the value is AND'ed with the *number* before any comparisons are done. The *number* is specified in C form. For instance, **13** is decimal, **013** is octal, and **0x13** is hexadecimal.

value The value to be compared with the value from the file. If the type is numeric, this value is specified in **C** form. If it is a string, it is specified as a **C** string with the usual escapes permitted (for instance, \n for NEWLINE).

Numeric values may be preceded by a character indicating the operation to be performed. It may be '=', to specify that the value from the file must equal the specified value, '<', to specify that the value from the file must be less than the specified value, '>', to specify that the value from the file must be greater than the specified value, '**&**', to specify that all the bits in the specified value must be set in the value from the file, '^', to specify that at least one of the bits in the specified value must not be set in the value from the file, or **x** to specify that any value will match. If the character is omitted, it is assumed to be '='.

For string values, the byte string from the file must match the specified byte string. The byte string from the file which is matched is the same length as the specified byte string.

message The message to be printed if the comparison succeeds. If the string contains a **printf**(3V) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character '>' indicates additional tests and messages to be printed. If the test on the line preceding the first line with a '>' succeeds, the tests specified in all the subsequent lines beginning with '>' are performed, and the messages printed if the tests succeed. The next line which does not begin with a '>' terminates this.

FILES

/etc/magic

SEE ALSO

file(1), **printf**(3V)

BUGS

There should be more than one level of subtests, with the level indicated by the number of '>' at the beginning of the line.

NAME

mtab – mounted file system table

SYNOPSIS

/etc/mtab

#include <mntent.h>

DESCRIPTION

mtab resides in the **/etc** directory, and contains a table of filesystems currently mounted by the **mount**(8) command. **umount** removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of **/etc/fstab**, described in **fstab**(5). There are a number of lines of the form:

fsname dir type opts freq passno

for example:

/dev/xy0a / 4.2 rw,noquota 1 2

The file is accessed by programs using **getmntent**(3), and by the system administrator using a text editor.

FILES

/etc/mtab
/etc/fstab

SEE ALSO

getmntent(3), **fstab**(5), **mount**(8)

NAME

netgroup – list of network groups

DESCRIPTION

netgroup defines network wide groups, used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in **netgroup** is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the **netgroup** file defines a group and has the format

groupname list-of-members

where members is either another group name, or a triple:

(*hostname*, *username*, *domainname*)

Any of these three fields can be empty, in which case it signifies a wild card. Thus

universal (, ,)

defines a group to which everyone belongs.

The *domainname* field must either be the local domain name or empty for the netgroup entry to be used. This field does *not* limit the netgroup or provide security. The *domainname* field refers to the domain in which the triple is valid, not the domain containing the trusted host.

A gateway machine should be listed under all possible hostnames by which it may be recognized:

wan (gateway, ,) (gateway-ebb, ,)

Field names that begin with something other than a letter, digit or underscore (such as '–') work in precisely the opposite fashion. For example, consider the following entries:

```
justmachines    (analytica,–,sun)
justpeople      (–,babbage,sun)
```

The machine **analytica** belongs to the group **justmachines** in the domain **sun**, but no users belong to it. Similarly, the user **babbage** belongs to the group **justpeople** in the domain **sun**, but no machines belong to it.

SEE ALSO

getnetgrent(3N), **exports**(5), **makedbm**(8), **ypserv**(8)

WARNINGS

The triple, (, , *domain*), allows all users and machines trusted access, and has the same effect as the triple, (, ,).

To correctly restrict access to a specific set of members, use the *hostname* and *username* fields of the triple.

NAME

netmasks – network mask data base

DESCRIPTION

The **netmasks** file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' notation (like IP host addresses, but with zeroes for the host part). For example,

128.32.0.0 255.255.255.0

can be used to specify the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field. When running the Network Information Service (NIS), this file on the master is used for the **netmasks.byaddr** map.

FILES

/etc/netmasks

SEE ALSO

ifconfig(8C)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

netrc – file for ftp remote login data

DESCRIPTION

The **.netrc** file contains data for logging in to a remote host over the network for file transfers by **ftp**(1C). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others (see **chmod**(1V)).

The following tokens are recognized; they may be separated by SPACE, TAB, or NEWLINE characters:

machine*name*
: Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the EOF is reached or another **machine** token is encountered.

login *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
: Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the **.netrc** file, **ftp** will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

account *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*
: Define a macro. This token functions as the **ftp macdef** command functions. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a null line (consecutive NEWLINE characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

EXAMPLE

The command:

machine ray login demo password mypassword

allows an autologin to the machine **ray** using the login name **demo** with password **mypassword**.

FILES

~/.netrc

SEE ALSO

chmod(1V), **ftp**(1C), **ftpd**(8C)

NAME

networks – network name data base

DESCRIPTION

The **networks** file contains information regarding the known networks which comprise the TCP/IP. For each network a single line should be present with the following information:

official-network-name network-number aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional '.' notation using the **inet_network ( )** routine from the Internet address manipulation library, **inet**(3N). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N), **inet**(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

orgrc – organizer configuration and initialization file

AVAILABILITY

Sun386i systems only.

DESCRIPTION

organizer(1) is a SunView 1 application for viewing and manipulating files and directories. It saves its parameters in the **.orgrc** file between runs. The user can use this file to configure **organizer**.

The first parameter in the file should always be the version number.

Version = 1.1

Change the version number only when necessary; if **organizer** determines that this version is "old", then it will save this version in **~/.orgrc.old** and try to copy **/usr/lib/Orgrc** into **~/.orgrc**.

The next two parameters assign default names for the system DOS Program and the default text editor.

DOS Program = dos
Text Editor = textedit

The DOS Program parameter should not be changed. However, the user can change the default text editor. For example:

Text Editor = shelltool vi

The Properties section initializes or customizes certain properties. The possible values for each item are listed below. The braces and vertical bars below indicate choices, they are not used in the **.orgrc** file. The **Update Interval** is in seconds.

 # Properties
PROPERTY Display Style = {Name and Icon | Name Only | Name and Info}
PROPERTY Roadmap = {Yes | No}
PROPERTY Show Hidden Files = {Yes | No}
PROPERTY Sort Type = {Name | File Type | Size | Date}
PROPERTY Sort Direction = {Ascending | Descending}
PROPERTY Update Interval = [5-300]

The Color Palette specifies all the color values used by **organizer**'s buttons and icons. These values must be RGB triplets. It is listed below.

Begin Color Palette
Background Color = 255, 255, 255
Directory Name Color = 0, 146, 236
Directory Icon Foreground Color = 114, 45, 0
Directory Icon Background Color = 255, 227, 185
Directory Highlight Name Color = 255, 255, 255
Text Name Color = 0, 166, 143
Text Icon Foreground Color = 0, 0, 0
Text Icon Background Color = 255, 255, 255
Text Highlight Name Color = 255, 255, 255
Executable Name Color = 255, 0, 0
Executable Icon Foreground Color = 157, 162, 187
Executable Icon Background Color = 255, 255, 255
Executable Highlight Name Color = 255, 255, 255
Device Name Color = 113, 117, 135
Device Icon Foreground Color = 0, 0, 0
Device Icon Background Color = 174, 255, 159
Device Highlight Name Color = 255, 255, 255
Button Group1 Color = 255, 220, 187

```
        Button Group2 Color = 201, 211, 232
        Button Group3 Color = 255, 244, 113
        Button Foreground Color = 0, 0, 0
        Button Background Color = 255, 255, 255
        Button Shadow Color = 180, 180, 184
        Button Highlight Color = 0, 0, 0
        Scrollbar Color = 142, 106, 146
    End Color Palette
```

The Color Labels section allows the labelling or "aliasing" of RGB triplets. The right side of a label assignment can contain an RGB triplet, a palette entry, or another label that has already been assigned. Here's an example:

```
    Begin Color Labels
        Black = Text Icon Foreground Color
        White = Background Color
        Orange = 255, 213, 127
        Dark Red = 232, 0, 0
        Steel Blue = 114, 146, 161
        Rasberry (sic) = 202, 140, 156
        Dark Blue = 0, 75, 161
        Light Gray = 223, 223, 223
        Maroon = 182, 84, 106
    End Color Labels
```

The rest of the **.orgrc** file contains user defined file types. The user can specify that certain files be grouped together and treated in a similar fashion. That is, the same icon is used to display all files in a file type, and the same command is used when a file is opened or edited. In the default **.orgrc** (**/usr/lib/Orgrc**) there are ten user defined file types. Here is an example of a user defined file type:

```
    Begin File Type Definition
        Name = *.c
        Background Icon = /usr/include/images/cMask.icon
        Foreground Icon = /usr/include/images/cStencil.icon
        Name Color = Black
        Icon Background Color = Orange
        Icon Foreground Color = Black
        Highlight Name Color = White
        Execute Application = cmdtool vi "$(FILE)"
        Edit Application = cmdtool vi "$(FILE)"
        Print Application = pr -f "$(FILE)" | lpr
    End File Type Definition
```

The right side of the **Name** field can contain any combination of **csh**(1) **Filename Substitution** characters. This field specifies the file type by way of its name. The next six fields together specify an **organizer** icon. This model allows a rich variety of icons. For more information, see the *Sun386i Advanced Skills* manual. The right side of the **Execute Application** entry specifies the command to execute when the user either opens or double clicks on a file of that type. The **Edit Application** and **Print Application** entries specify the command to execute when the user requests that a file of that type be edited or printed.

FILES

~/.orgrc	read at beginning of execution by the Organizer
/usr/lib/Orgrc	default **.orgrc** file

SEE ALSO

organizer(1)

Sun386i User's Guide
Sun386i Advanced Skills

LIMITATIONS

The right side of Color Palette entries must be RGB triplets.

Forward references for Color Labels are not allowed.

BUGS

organizer saves its parameters as it exits; unfortunately, it does not know how to save user's comments in the file. So, comments are blown away.

NAME

passwd – password file

SYNOPSIS

/etc/passwd

DESCRIPTION

The **passwd** file contains basic information about each user's account. This file contains a one-line entry for each authorized user, of the form:

username:*password*:*uid*:*gid*:*gcos-field*:*home-dir*:*login-shell*

where

username	is the user's login name. This field contains no uppercase characters, and must not be more than eight characters in length.
password	is the user's encrypted password, or a string of the form: *##name* if the encrypted password is in the **/etc/security/passwd.adjunct** file (see **passwd.adjunct**(5)). If this field is empty, **login**(1) does not request a password before logging the user in.
uid	is the user's numerical ID for the system, which must be unique. *uid* is generally a value between 0 and 32767.
gid	is the numerical ID of the group that the user belongs to. *gid* is generally a value between 0 an 32767.
gcos-field	is the user's real name, along with information to pass along in a mail-message heading. It is called the gcos-field for historical reasons. A **&** in this field stands for the login name (in cases where the login name appears in a user's real name).
home-dir	is the pathname to the directory in which the user is initially positioned upon logging in.
login-shell	is the user's initial shell program. If this field is empty, the default shell is **/usr/bin/sh**.

The **passwd** file can also have lines beginning with a '+' (plus sign) which means to incorporate entries from the Network Information Service (NIS). There are three styles of + entries in this file: by itself, + means to insert the entire contents of the NIS password file at that point; *+name* means to insert the entry (if any) for *name* from the NIS service at that point; *+@netgroup* means to insert the entries for all members of the network group **netgroup** at that point. If a *+name* entry has a non-null *password*, *gcos*, *home-dir*, or *login-shell* field, the value of that field overrides what is contained in the NIS service. The *uid* and *gid* fields cannot be overridden.

The **passwd** file can also have lines beginning with a '–' (minus sign) which means to disallow entries from the NIS service. There are two styles of '–' entries in this file: *–name* means to disallow any subsequent entries (if any) for *name* (in this file or in the NIS service); *–@netgroup* means to disallow any subsequent entries for all members of the network group *netgroup*.

The password file is an ASCII file that resides in the **/etc** directory. Because the encrypted passwords on a secure system are kept in the **passwd.adjunct** file, **/etc/passwd** has general read permission on all systems, and can be used by routines that map numerical user IDs to names.

Appropriate precautions must be taken to lock the **/etc/passwd** file against simultaneous changes if it is to be edited with a text editor; **vipw**(8) does the necessary locking.

EXAMPLE

Here is a sample **passwd** file when **passwd.adjunct** does not exist:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
fred:6k/7KCFRPNVXg:508:10:% Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

Here is a sample **passwd** file when **passwd.adjunct** does exist:

root:##root:0:10:God:/:/bin/csh
fred:##fred:508:10:& Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest

In this example, there are specific entries for users **root** and **fred**, to assure that they can log in even when the system is running standalone. The user **john** will have his password entry in the NIS service incorporated without change; anyone in the netgroup **documentation** will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a gcos-field of **Guest.**

FILES

/etc/passwd
/etc/security/passwd.adjunct

SEE ALSO

login(1), **mail**(1), **passwd**(1), **crypt**(3), **getpwent**(3V), **group**(5), **passwd.adjunct**(5), **adduser**(8), **sendmail**(8), **vipw**(8)

BUGS

mail(1) and **sendmail**(8) use the gcos-field to compose the **From:** line for addressing mail messages, but these programs get confused by nested parentheses when composing replies. This problem can be avoided by using different types of brackets within the gcos-field; for example:

(& Fredricks [Podunk U <EE/CIS>] {818}-555-5555)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

passwd.adjunct – user security data file

SYNOPSIS

/etc/security/passwd.adjunct

DESCRIPTION

The **passwd.adjunct** file contains the following information for each user:

name:*password*:*min-label*:*max-label*:*default-label*:*always-audit-flags*:*never-audit-flags*:

name	The user's login name in the system and it must be unique.
password	The encrypted password.
min-label	The lowest security level at which this user is allowed to login (not used at C2 level).
max-label	The highest security level at which this user is allowed to login (not used at C2 level).
default-label	The security level at which this user will run unless a label is specified at login.
always-audit-flags	Flags specifying events always to be audited for this user's processes; see **audit_control**(5).
never-audit-flags	Flags specifying events never to be audited for this user's processes; see **audit_control**(5).

Field are separated by a colon, and each user from the next by a NEWLINE.

The **passwd.adjunct** file can also have line beginning with a '+' (plus sign), which means to incorporate entries from the Network Information Service (NIS). There are three styles of '+' entries: all by itself, '+' means to insert the entire contents of the NIS **passwd.adjunct** file at that point; +*name* means to insert the entry (if any) for *name* from the NIS service at that point; +@*name* means to insert the entries for all members of the network group *name* at that point. If a '+' entry has a non-null password, it will override what is contained in the NIS service.

EXAMPLE

Here is a sample **/etc/security/passwd.adjunct** file:

```
root:q.mJzTnu8icF.:::::
ignatz:7KsI8CFRPNVXg::b,ap,bp,gp,dp,ic,r,d,l::+dc,+da:-dr:
rex:7HU8UUGRPNVXg:b,ap:b,ap,bp:b,bp::+ad:
+fred:9x.FFUw6xcJBa:::::
+:
```

The user **root** is the super-user, who has no special label constraints nor audit interest. The user **ignatz** may have any label from the lowest to the level **b** and any of a large number of categories. **ignatz** will run at system low unless he specifies otherwise. He is being audited on the system default event classes as well as data creations and access changes, but never for failed data reads. The user **rex** can function only at the level **b** and only in the categories **ap** or **ap** and **bp**. By default, he will run at '**b,bp**'. He is audited with the system defaults, except that successful administrative operations are not audited. The user **fred** will have the labels and audit flags that are specified in the NIS **passwd.adjunct** file. Any other users specified in the NIS service will be able to log in on this system.

The user security data file resides in the **/etc/security** directory. Because it contains encrypted passwords, it does not have general read permission.

FILES

/etc/security/passwd.adjunct
/etc/security

SEE ALSO

login(1), **passwd**(1), **crypt**(3), **getpwaent**(3), **getpwent**(3V), **audit_control**(5), **passwd**(5), **adduser**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

phones – remote host phone number data base

SYNOPSIS

/etc/phones

DESCRIPTION

The file **/etc/phones** contains the system-wide private phone numbers for the **tip**(1C) program. **/etc/phones** is normally unreadable, and so may contain privileged information. The format of **/etc/phones** is a series of lines of the form:

<system-name>[\t]*<phone-number>.

The system name is one of those defined in the **remote**(5) file and the phone number is constructed from **[0123456789–=*%]**. The '=' and '*' characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The '=' is required by the DF02-AC and the '*' is required by the BIZCOMP 1030.

Comment lines are lines containing a '#' sign in the first column of the line.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name **tip**(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1C), **remote**(5)

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in **plot**(3X), and are interpreted for various devices by commands described in **plot**(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an *l*, *m*, *n*, or *p* instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in **plot**(3X).

m Move: the next four bytes give a new current point.

n Cont: draw a line from the current point to the point given by the next four bytes. See **plot**(1G).

p Point: plot the point given by the next four bytes.

l Line: draw a line from the point given by the next four bytes to the point given by the following four bytes.

t Label: place the following ASCII string so that its first character falls on the current point. The string is terminated by a NEWLINE.

a Arc: the first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c Circle: the first four bytes give the center of the circle, the next two the radius.

e Erase: start another frame of output.

f Linemod: take the following string, up to a NEWLINE, as the style for drawing further lines. The styles are "dotted," "solid," "longdashed," "shortdashed," and "dotdashed." Effective only in **plot 4014** and **plot ver**.

s Space: the next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of **plot**(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

4014	**space(0, 0, 3120, 3120);**
ver	**space(0, 0, 2048, 2048);**
300, 300s	**space(0, 0, 4096, 4096);**
450	**space(0, 0, 4096, 4096);**

SEE ALSO

graph(1G), **plot**(1G), **plot**(3X)

NAME

pnp.sysnames – file used to allocate system names

SYNOPSIS

/etc/pnp.sysnames

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/pnp.sysnames** file contains system names that may be allocated on demand, typically as part of Automatic System Installation.

The system names should be legal system names, one per line. Legal names are up to 31 characters long, and consist of lowercase alphanumeric characters, dashes, and underscores. The first character must be alphabetic, and the last character should be alphanumeric. Blank lines are allowed in the file, but comments are not.

When a system name needs to be allocated, the first unused system name is taken from **/etc/pnp.sysnames**. If all the system names there are in use, unused names are allocated from the list *system-1*, *system-2*, . . . ; the default prefix *system* may be changed in the **/var/yp/updaters** makefile. A system name is "used" if there is already a matching entry in the Network Information Service (NIS) *hosts.byname* map, the *ethers.byname* map, or there is a netgroup with that name. Names are allocated to correspond to a given Ethernet address. There is no concept of "transient" name allocation; part of allocating a system name includes updating the *ethers.byname* and *ethers.byaddr* NIS maps to persistently associate the name with that Ethernet address.

One way to allocate a system name is to issue a **ypupdate**(3N) call to update the *ethers.byaddr* map. The key is the Ethernet address (or general IEEE 802.2 48 bit address, used also with FDDI and Token Ring standards) of the system whose name is being allocated. The data is a line formatted according to the format specified in **ethers**(5). A name is allocated if the name passed is '*' (a single asterisk). Updating this NIS map using **ypupdate**(3N) is a privileged operation, and may be performed only by users in the *networks* group (with group ID 12), or boot servers (listed in the *ypservers* NIS map).

FILES

/etc/pnp.sysnames
/usr/etc/yp/upd.systems
/var/yp/updaters

SEE ALSO

ypupdate(3N), **ethers**(5), **group**(5), **hosts**(5), **netgroup**(5), **updaters**(5), **pnpd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

policies – network administration policies

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **policies** file contains information relevant to domain-wide administration policies. Each line contains two tokens, separated by white space; the first token is the name of an administrative policy, and the second is the value of that policy.

FILES

/etc/policies
/var/yp/*domainname***/policies.{dir,pag}**

SEE ALSO

pnpd(8C), **rarpd**(8C), **logintool**(8)

NAME

printcap – printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

printcap is a simplified version of the **termcap**(5) data base for describing printers. The spooling system accesses the **printcap** file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base describes one printer. This data base may not be substituted for, as is possible for **termcap**, because it may allow accounting to be bypassed.

The default printer is normally **lp**, though the environment variable PRINTER may be used to override this. Each spooling utility supports a **–P***printer* option to explicitly name a destination printer.

Refer to *System and Network Administration* for a discussion of how to set up the database for a given printer. On Sun386i systems, refer to **snap**(1) for information on setting up printers with the system and network administration program.

Each entry in the **printcap** file describes a printer, and is a line consisting of a number of fields separated by ':' characters. The first entry for each printer gives the names which are known for the printer, separated by '|' characters. The first name is conventionally a number. The second name given is the most common abbreviation for the printer, and the last name given should be a long name fully identifying the printer. The second name should contain no blanks; the last name may well contain blanks for readability. Entries may continue onto multiple lines by giving a '\' as the last character of a line, and empty fields may be included for readability.

Capabilities in **printcap** are all introduced by two-character codes, and are of three types:

Boolean Capabilities that indicate that the printer has some particular feature. Boolean capabilities are simply written between the ':' characters, and are indicated by the word 'bool' in the **type** column of the capabilities table below.

Numeric Capabilities that supply information such as baud-rates, number of lines per page, and so on. Numeric capabilities are indicated by the word **num** in the **type** column of the capabilities table below. Numeric capabilities are given by the two-character capability code followed by the '#' character, followed by the numeric value. The following example is a numeric entry stating that this printer should run at 1200 baud:

:br#1200:

String Capabilities that give a sequence which can be used to perform particular printer operations such as cursor motion. String valued capabilities are indicated by the word **str** in the **type** column of the capabilities table below. String valued capabilities are given by the two-character capability code followed by an '=' sign and then a string ending at the next following ':'. For example,

:rp=spinwriter:

is a sample entry stating that the remote printer is named **spinwriter**.

Sun386i DESCRIPTION

On Sun386i systems, **lpr**(1) and related printing commands use the Network Information Service (NIS) to obtain the **printcap** entry for a named printer if the entry does not exist in the local **/etc/printcap** file. For example, when a user issues the command:

lpr -Pnewprinter foo

lpr searches **/etc/printcap** on the local system for an entry for **newprinter**. If no local entry for **newprinter** exists, then **lpr** searches the NIS map called **printcap**. The search is invisible to the user.

lpr creates the spooling directory for the printer automatically if no spooling directory exists.

System administrators can make a printer available to the entire NIS domain by placing an entry for that printer in the NIS **printcap** map, typically using **snap**. Otherwise, the system administrator must edit the **/etc/printcap** file on the NIS master and then rebuild the NIS map.

CAPABILITIES

Name	*Type*	*Default*	*Description*
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	TeX data filter (DVI format)
du	str	0	User ID of user 'daemon'.
fc	num	0	if lp is a tty, clear flag bits
ff	str	"\f"	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits
gf	str	NULL	graph data filter (plot(3X) format)
hl	bool	false	print the burst header page last
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of input/communication filter (created per job)
lf	str	"/dev/console"	error logging file name
lo	str	"lock"	name of lock file
lp	str	"/dev/lp"	device name to open for output
mc	num	0	maximum number of copies
ms	str	NULL	list of terminal modes to set or clear
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output/banner filter (created once)
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rg	str	NULL	restricted group. Only members of group allowed access
rm	str	NULL	machine name for remote printer
rp	str	"lp"	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open printer device read/write instead of write-only
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	"/var/spool/lpd"	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
st	str	"status"	status file name
tc	str	NULL	name of similar printer; must be last
tf	str	NULL	troff data filter (C/A/T phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits
xs	num	0	like 'xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

Note: the **fs**, **fc**, **xs**, and **xc** fields are flag *masks* rather than flag *values*. Certain default device flags are set when the device is opened by the line printer daemon if the device is connected to a terminal port. The flags indicated in the **fc** field are then cleared; the flags in the **fs** field are then set (or vice-versa, depending on the order of **fc#***nnnn* and **fs#***nnnn* in the **/etc/printcap** file). The bits cleared by the **fc** field and set by the **fs** field are those in the **sg_flags** field of the **sgtty** structure, as set by the **TIOCSETP ioctl** call, and the bits cleared by the **xc** field and set by the **xs** field are those in the "local flags" word, as set by the **TIOCLSET ioctl** call. See **ttcompat**(4M) for a description of these flags. For example, to set exactly the flags 06300 in the **fs** field, which specifies that the **EVENP**, **ODDP**, and **XTABS** modes are to be set, and all other flags are to be cleared, do:

:fc#0177777:fs#06300:

The same process applies to the **xc** and **xs** fields. Alternatively, the **ms** field can be used to specify modes to be set and cleared. These modes are specified as **stty**(1V) modes; any mode supported by **stty** may be specified, except for the baud rate which must be specified with the **br** field. This permits modes not supported by the older terminal interface described in **ttcompat**(4M) to be set or cleared. Thus, to set the terminal port to which the printer is attached to even parity, TAB expansion, no NEWLINE to RETURN/LINEFEED translation, and RTS/CTS flow control enabled, do:

:ms=evenp,-tabs,nl,crtscts:

On Sun386i systems, the **tc** field, as in the **termcap**(5) file, must appear last in the list of capabilities. It is recommended that each type of printer have a general entry describing common capabilities; then an individual printer can be defined with its particular capabilities plus a **tc** field that points to the general entry for that type of printer.

FILES

/etc/printcap

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **plot**(1G), **snap**(1), **stty**(1V), **plot**(3X), **ttcompat**(4M), **termcap**(5), **lpc**(8), **lpd**(8), **pac**(8)

System and Network Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

proto – prototype job file for at

SYNOPSIS

/var/spool/cron/.proto

/var/spool/cron/.proto.*queue*

DESCRIPTION

When a job is submitted to **at** or **batch**, (see **at**(1)) the job is constructed as a shell script. First, a prologue is constructed, consisting of:

- A header specifying the owner, job name, and shell that should be used to run the job, and a flag indicating whether mail should be sent when the job completes;
- A set of Bourne shell commands to make the environment (see **environ**(5V)) for the **at** job the same as the current environment;
- A command to run the user's shell (as specified by the **SHELL** environment variable) with the rest of the job file as input.

at then reads a "prototype file," and constructs the rest of the job file from it.

Text from the prototype file is copied to the job file, except for special "variables" that are replaced by other text:

$d	is replaced by the current working directory
$l	is replaced by the current file size limit (see **ulimit**(3C))
$m	is replaced by the current umask (see **umask**(2V))
$t	is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon
$<	is replaced by text read by **at** from the standard input (that is, the commands provided to **at** to be run in the job)

If the job is submitted in queue *queue*, **at** uses the file **/var/spool/cron/.proto.***queue* as the prototype file if it exists, otherwise it will use the file **/var/spool/cron/.proto**.

EXAMPLES

The standard **.proto** file supplied with SunOS is:

```
#
# @(#)proto.5 1.3 89/10/05 SMI; from S5R3 1.1
#
cd $d
umask $m
$<
```

which causes commands to change the current directory in the job to the current directory at the time **at** was run, and to change the umask in the job to the umask at the time **at** was run, to be inserted before the commands in the job.

FILES

/var/spool/cron/.proto
/var/spool/cron/.proto.*queue*

SEE ALSO

at(1)

NAME

protocols – protocol name data base

SYNOPSIS

/etc/protocols

DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the TCP/IP. For each protocol a single line should be present with the following information:

official-protocol-name protocol-number aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLE

The following example is taken from SunOS.

```
#
# Internet (IP) protocols
#
ip      0   IP      # internet protocol, pseudo protocol number
icmp    1   ICMP    # internet control message protocol
ggp     3   GGP     # gateway-gateway protocol
tcp     6   TCP     # transmission control protocol
pup     12  PUP     # PARC universal packet protocol
udp     17  UDP     # user datagram protocol
```

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

publickey – public key database

SYNOPSIS

/etc/publickey

DESCRIPTION

/etc/publickey is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with its login password (also in hex notation).

This file is altered either by the user through the **chkey**(1) command or by the system administrator through the **newkey**(8) command. The file **/etc/publickey** should only contain data on the Network Information Service (NIS) master machine, where it is converted into the NIS database **publickey.byname**.

The **/etc/publickey** file contains a default entry for **nobody**. If this entry is commented out, **chkey** only allows user to edit their existing entry, it will not allow them to create new entries.

SEE ALSO

chkey(1), **publickey**(3R), **newkey**(8), **ypupdated**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

queuedefs – queue description file for at, batch, and cron

SYNOPSIS

/var/spool/cron/queuedefs

DESCRIPTION

The **queuedefs** file describes the characteristics of the queues managed by **cron**(8). Each non-comment line in this file describes one queue. The format of the lines are as follows:

q.[*njob***j**][*nice***n**][*nwait***w**]

The fields in this line are:

q The name of the queue. **a** is the default queue for jobs started by **at**(1); **b** is the default queue for jobs started by **batch** (see **at**(1)); **c** is the default queue for jobs run from a **crontab**(5) file.

njob The maximum number of jobs that can be run simultaneously in that queue; if more than *njob* jobs are ready to run, only the first *njob* jobs will be run, and the others will be run as jobs that are currently running terminate. The default value is 100.

nice The **nice**(1) value to give to all jobs in that queue that are not run with a user ID of super-user. The default value is 2.

nwait The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because more than 25 jobs were running in all the queues. The default value is 60.

Lines beginning with # are comments, and are ignored.

EXAMPLE

```
#
# @(#)queuedefs 1.1 87/02/18 SMI; from S5R3
#
a.4j1n
b.2j2n90w
```

This file specifies that the **a** queue, for **at** jobs, can have up to 4 jobs running simultaneously; those jobs will be run with a **nice** value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it. The **b** queue, for **batch** jobs, can have up to 2 jobs running simultaneously; those jobs will be run with a **nice** value of 2. If a job cannot be run because too many other jobs are running, **cron** will wait 90 seconds before trying again to run it. All other queues can have up to 100 jobs running simultaneously; they will be run with a **nice** value of 2, and if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it.

FILES

/var/spool/cron/queuedefs

SEE ALSO

at(1), **nice**(1), **crontab**(5), **cron**(8)

NAME

rasterfile – Sun's file format for raster images

SYNOPSIS

#include <rasterfile.h>

DESCRIPTION

A rasterfile is composed of three parts: first, a header containing 8 integers; second, a (possibly empty) set of colormap values; and third, the pixel image, stored a line at a time, in increasing *y* order. The image is layed out in the file as in a memory pixrect. Each line of the image is rounded up to the nearest 16 bits.

The header is defined by the following structure:

```
struct rasterfile {
        int     ras_magic;
        int     ras_width;
        int     ras_height;
        int     ras_depth;
        int     ras_length;
        int     ras_type;
        int     ras_maptype;
        int     ras_maplength;
};
```

The *ras_magic* field always contains the following constant:

```
#define RAS_MAGIC    0x59a66a95
```

The *ras_width*, *ras_height*, and *ras_depth* fields contain the image's width and height in pixels, and its depth in bits per pixel, respectively. The depth is either 1 or 8, corresponding to standard frame buffer depths. The *ras_length* field contains the length in bytes of the image data. For an unencoded image, this number is computable from the *ras_width*, *ras_height*, and *ras_depth* fields, but for an encoded image it must be explicitly stored in order to be available without decoding the image itself. Note: the length of the header and of the (possibly empty) colormap values are not included in the value of the *ras_length* field; it is only the image data length. For historical reasons, files of type RT_OLD will usually have a 0 in the *ras_length* field, and software expecting to encounter such files should be prepared to compute the actual image data length if needed. The *ras_maptype* and *ras_maplength* fields contain the type and length in bytes of the colormap values, respectively. If *ras_maptype* is not RMT_NONE and the *ras_maplength* is not 0, then the colormap values are the *ras_maplength* bytes immediately after the header. These values are either uninterpreted bytes (usually with the *ras_maptype* set to RMT_RAW) or the equal length red, green and blue vectors, in that order (when the *ras_maptype* is RMT_EQUAL_RGB). In the latter case, the *ras_maplength* must be three times the size in bytes of any one of the vectors.

SEE ALSO

SunView Programmer's Guide

NAME

remote – remote host description file

SYNOPSIS

/etc/remote

DESCRIPTION

The systems known by **tip**(1C) and their attributes are stored in an ASCII file which is structured somewhat like the **termcap**(5) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon '**:**'. Lines ending in a '\' character with an immediately following NEWLINE are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '=' sign indicates a string value follows. A field name followed by a '#' sign indicates a following numeric value.

Entries named **tip***baudrate* are used as default entries by **tip**, as follows. When **tip** is invoked with only a phone number, it looks for an entry of the form **tip***baudrate*, where *baudrate* is the baud rate with which the connection is to be made. For example, if the connection is to be made at 300 baud, **tip** looks for an entry of the form **tip300**.

CAPABILITIES

Capabilities are either strings **(str)**, numbers **(num)**, or boolean flags **(bool)**. A string capability is specified by *capability=value*; for example, '**dv=/dev/harris**'. A numeric capability is specified by *capability#value*; for example, '**xa#99**'. A boolean capability is specified by simply listing the capability.

at **(str)** Auto call unit type. The following lists valid '**at**' types and their corresponding hardware:

biz31f	Bizcomp 1031, tone dialing
biz31w	Bizcomp 1031, pulse dialing
biz22f	Bizcomp 1022, tone dialing
biz22w	Bizcomp 1022, pulse dialing
df02	DEC DF02
df03	DEC DF03
ventel	Ventel 212+
v3451	Vadic 3451 Modem
v831	Vadic 831
hayes	Any Hayes-compatible modem
at	Any Hayes-compatible modem

br **(num)** The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.

cm **(str)** An initial connection message to be sent to the remote host. For example, if a host is reached through a port selector, this might be set to the appropriate sequence required to switch to the host.

cu **(str)** Call unit if making a phone call. Default is the same as the **dv** field.

di **(str)** Disconnect message sent to the host when a disconnect is requested by the user.

du **(bool)** This host is on a dial-up line.

dv **(str)** Device(s) to open to establish a connection. If this file refers to a terminal line, **tip** attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.

ec **(bool)** Initialize the **tip** variable **echocheck** to *on*, so that **tip** will synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted.

el **(str)** Characters marking an end-of-line. The default is no characters. **tip** only recognizes '~' escapes after one of the characters in **el**, or after a RETURN.

es **(str)** The command prefix (escape) character for **tip**.

et (**num**) Number of seconds to wait for an echo response when echo-check mode is on. This is a decimal number. The default value is 10 seconds.

ex (**str**) Set of non-printable characters not to be discarded when scripting with beautification turned on. The default value is '\t\n\b\f'.

fo (**str**) Character used to force literal data transmission. The default value is '\377'.

fs (**num**) Frame size for transfers. The default frame size is equal to 1024.

hd (**bool**) Initialize the **tip** variable **halfduplex** to *on*, so local echo should be performed.

hf (**bool**) Initialize the **tip** variable **hardwareflow** to *on*, so hardware flow control is used.

ie (**str**) Input end-of-file marks. The default is a null string ("").

nb (**bool**) Initialize the **tip** variable **beautify** to *off*, so that unprintable characters will not be discarded when scripting.

nt (**bool**) Initialize the **tip** variable **tandem** to *off*, so that XON/XOFF flow control will not be used to throttle data from the remote host.

nv (**bool**) Initialize the **tip** variable **verbose** to *off*, so that verbose mode will be turned on.

oe (**str**) Output end-of-file string. The default is a null string (""). When **tip** is transferring a file, this string is sent at end-of-file.

pa (**str**) The type of parity to use when sending data to the host. This may be one of **even**, **odd**, **none**, **zero** (always set bit 8 to zero), **one** (always set bit 8 to 1). The default is **none**.

pn (**str**) Telephone number(s) for this host. If the telephone number field contains an '@' sign, **tip** searches the **/etc/phones** file for a list of telephone numbers — see **phones**(5). A '%' sign in the telephone number indicates a 5-second delay for the Ventel Modem.

pr (**str**) Character that indicates end-of-line on the remote host. The default value is '\n'.

ra (**bool**) Initialize the **tip** variable **raise** to *on*, so that lower case letters are mapped to upper case before sending them to the remote host.

rc (**str**) Character that toggles case-mapping mode. The default value is '\377'.

re (**str**) The file in which to record session scripts. The default value is **tip.record**.

rw (**bool**) Initialize the **tip** variable **rawftp** to *on*, so that all characters will be sent as is during file transfers.

sc (**bool**) Initialize the **tip** variable **script** to *on*, so that everything transmitted by the remote host will be recorded.

tb (**bool**) Initialize the **tip** variable **tabexpand** to *on*, so that tabs will be expanded to spaces during file transfers.

tc (**str**) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
	:dv=/dev/cua0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
	:pn=7654321%:tc=UNIX-1200
```

FILES

/etc/remote
/etc/phones

SEE ALSO

tip(1C), **phones**(5), **termcap**(5)

NAME

resolv.conf – configuration file for domain name system resolver

DESCRIPTION

The resolver configuration file contains information that is read by the domain name system resolver library the first time it is invoked in a process. It is only necessary to create this file to specify an explicit default domain name other than the default one derived from the **domainname**(1) command, or to specify name servers to use on other machines. The file is designed to be human readable and contains a list of keyword-value pairs that provide various types of resolver information.

keyword value

The different configuration options are:

nameserver *address* The Internet address (in dot notation) of a name server that the resolver should query. Up to MAXNS (currently 3) name servers may be listed. In that case the resolver library queries tries them in the order listed. The policy used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made. If there are no **nameserver** lines in this file, then the loopback address is used, so there must be a name server running on the same machine.

domain *name* The default domain to append to names that do not have a dot in them, and used in searches. If there is no **domain** line in this file, then it is derived from the domain set by the **domainname**(1) command, usually by removing the first component. For example, if the **domainname**(1) is set to ''foo.podunk.edu'' then the default domain used by the resolver will be ''podunk.edu''. The is the same policy used by **sendmail**(8).

The keyword-value pair must appear on a single line, and the keyword (for instance, **nameserver**) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

domainname(1), **gethostent**(3N), **resolver**(3), **named**(8C), **nslookup**(8C), RFC 1034, RFC 1035

System and Network Administration

NAME

rfmaster – Remote File Sharing name server master file

SYNOPSIS

/usr/nserve/rfmaster

DESCRIPTION

The **rfmaster** file is an ASCII text file that identifies the hosts that are responsible for providing primary and secondary domain name service for Remote File Sharing domains. This file contains a series of entries, each terminated by a NEWLINE; a record may be extended over more than one line by escaping the NEWLINE with a backslash. Fields in each record are separated by white space. Each record has three fields: *name*, *type*, and *data*.

The *type* field, which defines the meaning of the *name* and *data* fields, has three possible values:

p Primary domain name server. In this case, *name* is the domain name and *data* is the full hostname of the primary name server, specified as:

domain.nodename

There can be only one primary name server per domain.

s Define a secondary name server for a domain. In this case, *name* and *data* are the same as for the **p** type. The order of the **s** entries in the **rfmaster** file determines the order in which secondary name servers take over when the current domain name server fails.

a Define a network address for a machine. In this case, *name* is the full domain name for the machine, and *data* is the network address. The network address can be in plain ASCII text or it can be preceded by a '**\x**' to be interpreted as hexadecimal notation.

There are at least two lines in the **rfmaster** file per domain name server: one **p** line and one **a** line. Together, they define the primary and its network address. There should also be at least one secondary name server in each domain.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If this file is missing, the **–p** option of **rfstart** must be used to identify the primary. After that, a copy of the primary's **rfmaster** file is automatically placed on the machine.

Domains not served by the primary can also be listed in the **rfmaster** file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary will be able to share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary.

EXAMPLE

An example of an **rfmaster** file is shown below. The network addresses given in the example are IP addresses; for more information on their format and how to generate them, see **hostrfs**(8).

```
sunrfs          p    sunrfs.estale
sunrfs          s    sunrfs.ivy
sunrfs.estale   a    \x000214508190320d
sunrfs.ivy      a    \x0002145081903246
```

Note: If a line in the **rfmaster** file begins with a '#' (pound sign) character, the entire line will be treated as a comment.

FILES

/usr/nserve/rfmaster

SEE ALSO

rfstart(8)

System and Network Administration

NAME

rgb – available colors (by name) for coloredit

SYNOPSIS

.rgb

$HOME/.rgb

/usr/lib/.rgb

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

.rgb is an ASCII file containing consecutive lines terminated by newlines. Each line starts with three integers, each in the range 0-255. These integers are the RGB equivalent for the color named on the same line. At least one tab character delimits the last integer from the name field. The coloreditor searches for this file, first in the current directory; next, in the users home directory; and finally, in **/usr/lib**. The user can add to or delete from the **.rgb** file that he or she has access to, thus changing the available color table for subsequent invocations of coloredit.

EXAMPLES

The following is an example of a **.rgb** file.

0 0 0	**Black**
0 0 255	**Blue**
95 159 159	**Cadet Blue**
66 66 111	**Cornflower Blue**
107 35 142	**Dark Slate Blue**

SEE ALSO

coloredit(1)

NAME

rootmenu – root menu specification for SunView

SYNOPSIS

~/.rootmenu
/usr/lib/.rootmenu

DESCRIPTION

If a **.rootmenu** file is present in a user's home directory, it specifies the SunView menu, the menu that appears when the user clicks and holds the right mouse button in the background of the SunView desktop. If a **.rootmenu** file is not present in the user's home directory, **/usr/lib/.rootmenu** specifies the SunView menu.

Each line of a **.rootmenu** file has the format:

menu item *command*

menu item can be a character string, or an icon file delimited by angle brackets:

<icon-filename>

If *menu item* is a character string with embedded spaces, it must be enclosed by double quotes ('"').

command can be a command line to be executed when the menu item is selected, or one of the following reserved-word commands:

EXIT Exit **sunview** (requires confirmation).

REFRESH Redraw the entire screen.

MENU This menu item is a pull-right item with a submenu. If a full pathname follows the **MENU** command, the submenu contents are taken from that file. Otherwise, all the lines between a **MENU** command and a matching **END** command are added to the submenu.

END Mark the end of a nested submenu. The left side of this line should match the left side of a line with a **MENU** command.

If *command* is not one of the reserved-word commands, it is treated as a command line, although no shell interpretation is done.

Lines beginning with a '#' character are considered comments and are ignored.

If a user's **.rootmenu** file is modified, the SunView menu immediately reflects the changes.

See **sunview**(1) for more details about **.rootmenu**.

EXAMPLES

The following is a sample **.rootmenu** file:

```
#
#       sample root menu
#
"Lock Screen"           lockscreen
Tools   MENU
        Perfmeter               perfmeter
        Calculator              calc
        Mailtool                mailtool
Tools   END
"ShellTool"             shelltool
"CommandTool"           cmdtool
"Console"               cmdtool -C
#"MailTool"             mailtool
"TextEditor"            textedit
```

"DefaultsEditor"	**defaultsedit**
#"IconEditor"	**iconedit**
#"DbxTool"	**dbxtool**
"PerfMeter"	**perfmeter**
#"GraphicsTool"	**gfxtool**
"Redisplay All"	**REFRESH**
"Exit Suntools"	**EXIT**

SEE ALSO

sunview(1)

NAME

rpc – rpc program number data base

SYNOPSIS

/etc/rpc

DESCRIPTION

The **rpc** file contains user readable names that can be used in place of rpc program numbers. Each line has the following information:

rpc-program-server *rpc-program-number* *aliases*

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

EXAMPLE

Here is an example of the **/etc/rpc** file from the SunOS System.

```
#
#       rpc  1.10  87/04/10
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
3270_mapper     100013
rje_mapper      100014
selection_svc   100015  selnsvc
database_svc    100016
rexd            100017  rex
alis            100018
sched           100019
llockmgr        100020
nlockmgr        100021
x25.inr         100022
statmon         100023
status          100024
bootparam       100026
ypupdated       100028  ypupdate
keyserv         100029  keyserver
```

FILES

/etc/rpc

SEE ALSO

getrpcent(3N)

NAME

sccsfile – format of an SCCS history file

DESCRIPTION

An SCCS file is an ASCII file consisting of six logical parts:

checksum	character count used for error detection
delta table	log containing version info and statistics about each delta
usernames	login names and/or group IDs of users who may add deltas
flags	definitions of internal keywords
comments	arbitrary descriptive information about the file
body	the actual text lines intermixed with control lines

Each section is described in detail below.

Conventions

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the *control character*, and will be represented as '^A'. If a line described below is not depicted as beginning with the control character, it cannot do so and still be within SCCS file format.

Entries of the form *ddddd* represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

^A **h***ddddd*

The value of the checksum is the sum of all characters, except those contained in the first line. The ^**Ah** provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

^As *inserted/deleted/unchanged*
^**Ad** *type sid yr/mo/da hr:mi:se username serial-number predecessor-sn*
^**Ai** *include-list*
^**Ax** *exclude-list*
^**Ag** *ignored-list*
^**Am** *mr-number*
. . .
^Ac *comments . . .*
. . .
^**Ae**

The first line (^**As**) contains the number of lines inserted/deleted/unchanged respectively. The second line (^**Ad**) contains the type of the delta (normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the user-name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The ^**Ai**, ^**Ax**, and ^**Ag** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines do not always appear.

The ^**Am** lines (optional) each contain one MR number associated with the delta; the ^**Ac** lines contain comments associated with the delta.

The ^**Ae** line ends the delta table entry.

User Names

The list of user-names and/or numerical group IDs of users who may add deltas to the file, separated by NEWLINE characters. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **^Au** and **^AU**. An empty list allows anyone to make a delta.

Flags

Flags are keywords that are used internally (see **sccs-admin**(1) for more information on their use). Each flag line takes the form:

^Af *flag optional text*

The following flags are defined in order of appearance:

^Af t *type-of-program*
: Defines the replacement for the **%T%** ID keyword.

^Af v program-name
: Controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program.

^Af i
: Indicates that the **'No id keywords'** message is to generate an error that terminates the SCCS command. Otherwise, the message is treated as a warning only.

^Af b
: Indicates that the **−b** option may be used with the SCCS **get** command to create a branch in the delta tree.

^Af m module name
: Defines the first choice for the replacement text of the **%M%** ID keyword.

^Af f *floor*
: Defines the "floor" release; the release below which no deltas may be added.

^Af c *ceiling*
: Defines the "ceiling" release; the release above which no deltas may be added.

^Af d *default-sid*
: The **d** flag defines the default SID to be used when none is specified on an SCCS **get** command.

^Af n
: The **n** flag enables the SCCS **delta** command to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped).

^Af j
: Enables the SCCS **get** command to allow concurrent edits of the same base SID.

^Af l *lock-releases*
: Defines a *list* of releases that are locked against editing.

^Af q user defined
: Defines the replacement for the **%Q%** ID keyword.

^Af e **0|1**
: The **e** flag indicates whether a source file is encoded or not. A **1** indicates that the file is encoded. Source files need to be encoded when they contain control characters, or when they do not end with a NEWLINE. The **e** flag allows files that contain binary data to be checked in.

Comments

Arbitrary text surrounded by the bracketing lines **^At** and **^AT**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

^AI *ddddd*
^AD *ddddd*
^AE *ddddd*

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **what**(1)

NAME

services – Internet services and aliases

DESCRIPTION

The **services** file contains an entry for each service available through the TCP/IP. Each entry consists of a line of the form:

service-name port / protocol aliases

service-name — This is the official Internet service name.

port / protocol — This field is composed of the port number and protocol through which the service is provided (for instance, **512/tcp**).

aliases — This is a list of alternate names by which the service might be requested.

Fields can be separated by any number of spaces or TAB's. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N), **inetd.conf**(5)

BUGS

A name server should be used instead of a static file.

NAME

setup.pc – master configuration file for DOS

SYNOPSIS

~/pc/setup.pc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **setup.pc** file in your home PC directory, **~/pc**, is the master configuration file for DOS. Changes to the file take effect for all new DOS windows you start. The definitions made in **setup.pc** and **AUTOEXEC.BAT** serve to define your system to DOS. Among other things, the **setup.pc** file defines:

- The printers or devices to which you assign DOS printer names (LPT1, LPT2, LPT3)
- The devices or boards that are tied to the DOS communications devices (COM1, COM2)
- The name of a special DOS quick-start file that you may have set up
- The drive C file to be used

The format of each line is as follows; separators can be TAB or SPACE characters:

DOS Device *SunOS Device or Command*

DOS Device
The name of the device as DOS knows it. For example, the device name for the first diskette drive in DOS is “A”.

SunOS Device or Command
The name of the device as the SunOS system knows it. This can also be a symbolic link to the real device name. For example, **/etc/dos/defaults/diskette_a** is a symbolic link to **/dev/rfd0c**. For emulated DOS printers (LPT1, LPT2, or LPT3), specify a command or command pipeline.

EXAMPLES

```
# DOS Device    SunOS Device Path Name
#
A               /etc/dos/defaults/diskette_a
#B              /etc/dos/defaults/diskette_b
C               ~/pc/C:
COM1            /etc/dos/defaults/com1
#COM2           /etc/dos/defaults/com2
LPT1            lpr
LPT2            cat >>~/lpt-2
LPT3            psfx80 | lpr
SAVE            ~/pc/.quickpc
#CMDTOOL
#TEXT
#BOARDS
```

Placed at the beginning of a line to indicate a comment.

A, B Diskette drivers defined using the standard SunOS names for the Sun386i diskette drives. Drive **A** is normally assigned to the built-in diskette drive.

C The emulated C drive. It is actually stored as one large system file.

COM1, COM2
Serial ports. The first DOS serial port (COM1) is assigned to the Sun386i built-in serial port. To use the built-in serial port as COM2, comment out the COM1 line and uncomment the COM2 line. (DOS Windows directs the output of either COM1 or COM2 to the built-in port, but uses different interrupt levels so that COM2 "appears" to DOS to be a second serial port.) You can also add a real second serial port by installing an AT or XT card and enabling the SunOS ATS driver.

LPT1, LPT2, LPT3
Emulated printers. DOS printer names can be assigned to SunOS printers, other devices, or files.

SAVE The "quick-start" file DOS reads at startup for faster loading.

CMDTOOL
Used to list the SunOS commands that must run in a separate Commands window when started from DOS. The following SunOS commands automatically run in a Commands window when you run them from DOS:

mail man more passwd rlogin stty vi

If there are other SunOS commands or applications you want to run from DOS, and these commands require keyboard entry or Commands window display, list them here. If you add entries to this line, separate them with a SPACE character, and be sure to remove the # (comment) symbol to activate the line.

TEXT Specifies a list of "text-only" DOS programs. Such programs do not require a PC window because they do not print at specific screen positions; they can print text in a current Commands window if that is where you are working at the time. An example is a C compiler or a linker that runs from the DOS command line. If you place entries on this line, separate them with a SPACE character, and be sure to remove the # symbol to activate the text-only line.

BOARDS
A list of boards that DOS should attempt to activate when opening a DOS window. Each board you list here must have a corresponding entry in the **boards.pc file** (see **boards.pc**(5)).

You can create task-specific DOS environments by setting up additional **setup.pc** files to attach different printers, drive C files, and other real and emulated devices.

If you are installing a board that duplicates a function normally enabled in the **setup.pc** file, you should disable the corresponding **setup.pc** line by commenting it out with #.

FILES

~/pc/setup.pc Personal **setup.pc** file, copied to the user's **pc** directory when DOS is started for the first time.

/etc/dos/defaults/setup.pc Master copy of **setup.pc** for the workstation.

SEE ALSO

dos(1), **boards.pc**(5)

Sun386i User's Guide,
Sun386i Advanced Skills,
Sun MS-DOS Reference Manual

NAME

sm, sm.bak, sm.state – in.statd directory and file structures

SYNOPSIS

/etc/sm, /etc/sm.bak, /etc/sm.state

DESCRIPTION

/etc/sm and **/etc/sm.bak** are directories generated by **in.statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **in.statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **in.statd** daemon upon its recovery.

/etc/sm.state is a file generated by **rpc.statd** to record the its version number. This version number is incremented each time a crash or recovery takes place.

FILES

/etc/sm
/etc/sm.bak
/etc/sm.state

SEE ALSO

lockd(8C), **statd**(8C)

NAME

sm, sm.bak, state – statd directories and file structures

SYNOPSIS

/etc/sm /etc/sm.bak /etc/state

DESCRIPTION

/etc/sm and **/etc/sm.bak** are directories generated by **statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **statd** daemon upon its recovery.

/etc/state is a file generated by **statd** to record its version number. This version number is incremented each time a crash or recovery takes place.

FILES

/etc/sm
/etc/sm.bak
/etc/state

SEE ALSO

lockd(8C), **statd**(8C)

NAME

sunview – initialization file for SunView

SYNOPSIS

~/.sunview
~/.suntools
/usr/lib/.sunview

DESCRIPTION

If the file **.sunview** or **.suntools** is present in a user's home directory when the user starts up **sunview**(1), **sunview** starts up the "tools", or window-based applications listed in this file. You can use a **.sunview** or **.suntools** file to customize your desktop layout. If a **.sunview** or **.suntools** file is not present in the user's home directory, **sunview** starts up the tools listed in **/usr/lib/.sunview**.

Each line of a **.sunview** or **.suntools** file has the format:

SunView-tool [*options*]

SunView-tool is in the form of a command line, although no shell interpretation is done. *options* are command line options which may include SunView generic tool arguments (see **sunview**(1) for a description of generic tool arguments). Lines beginning with the '#' character are considered comments and are ignored.

EXAMPLES

Here is a sample **.sunview** file:

```
#
#         sample .sunview file
#
cmdtool   -Wp   0   0 -WP    0   0 -Wh  3 -Ww 80 -Wl "<< CONSOLE >>" -WL "console" -C
clock     -Wp 497  32 -WP  704   0 -Wi -Wh 1
cmdtool   -Wp   0  71 -WP  772   0 -Wi -Wh 44 -Ww 80
textedit  -Wp 259  98 -WP  840   0 -Wi
mailtool  -Wp 492  71 -WP  908   0 -Wi
```

SEE ALSO

sunview(1), **toolplaces**(1)

NAME

svdtab – SunView device table

SYNOPSIS

/etc/svdtab

DESCRIPTION

The **/etc/svdtab** contains information that is used by the window system (for example, **sunview**(1)) to change the owner, group, and permissions of the window devices (**/dev/win***) upon startup. By default *all* lines in this file are commented out. That is, all security is disabled. To enable security, uncomment the following line in **/etc/svdtab** and start up the window system again:

#0600

If **/etc/svdtab** contains an entry, the owner and group of the **win** devices are set to the owner and group of the console. The permissions are set as specified in **/etc/svdtab**. The recommended permissions for normal security is **0600**.

Once the window devices are owned by the user, their permissions and ownership can be changed using **chmod**(1V) and **chown**(8), as with any user-other file.

EXAMPLES

The following is an example entry of the **/etc/svdtab** file:

0600

This entry specifies that upon SunView startup, the owner, group and permissions of **/dev/win*** will be set to the user's username, the user's group and **0600**, respectively. Upon exiting the window system, the owner and group of **/dev/win***, will be reset to **root**, and **wheel**. The permissions remain as set in **/etc/svdtab**. If no entry appears in this file, the owner, group and permissions will *not* be changed.

SEE ALSO

chmod(1V), **sunview**(1), **chown**(8)

NOTES

If the window system dies unnaturally, for example by **kill**(1), the owner, group and permissions remain as set when the window was started up.

NAME

syslog.conf – configuration file for syslogd system log daemon

SYNOPSIS

/etc/syslog.conf

DESCRIPTION

The file **/etc/syslog.conf** contains information used by the system log daemon, **syslogd**(8), to forward a system message to appropriate log files and/or users. **syslog** preprocesses this file through **m4**(1V) to obtain the correct information for certain log files.

A configuration entry is composed of two TAB-separated fields:

selector *action*

The *selector* field contains a semicolon-separated list of priority specifications of the form:

facility.*level*[;*facility*.*level*]

where *facility* is a system facility, or comma-separated list of facilities, and *level* is an indication of the severity of the condition being logged. Recognized values for *facility* include:

user Messages generated by user processes. This is the default priority for messages from programs or facilities not listed in this file.

kern Messages generated by the kernel.

mail The mail system.

daemon System daemons, such as **ftpd**(8C), **routed**(8C), etc.

auth The authorization system: **login**(1), **su**(1V), **getty**(8), etc.

lpr The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc.

news Reserved for the USENET network news system.

uucp Reserved for the UUCP system; it does not currently use the **syslog** mechanism.

cron The **cron/at** facility; **crontab**(1), **at**(1), **cron**(8), etc.

local0-7 Reserved for local use.

mark For timestamp messages produced internally by **syslogd**.

***** An asterisk indicates all facilities except for the **mark** facility.

Recognized values for *level* are (in descending order of severity):

emerg For panic conditions that would normally be broadcast to all users.

alert For conditions that should be corrected immediately, such as a corrupted system database.

crit For warnings about critical conditions, such as hard device errors.

err For other errors.

warning For warning messages.

notice For conditions that are not error conditions, but may require special handling.

info Informational messages.

debug For messages that are normally used only when debugging a program.

none Do not send messages from the indicated *facility* to the selected file. For example, a *selector* of

***.debug;mail.none**

will send all messages *except* mail messages to the selected file.

The *action* field indicates where to forward the message. Values for this field can have one of four forms:

- A filename, beginning with a leading slash, which indicates that messages specified by the *selector* are to be written to the specified file. The file will be opened in append mode.
- The name of a remote host, prefixed with an **@**, as with: **@***server*, which indicates that messages specified by the *selector* are to be forwarded to the **syslogd** on the named host.
- A comma-separated list of usernames, which indicates that messages specified by the *selector* are to be written to the named users if they are logged in.
- An asterisk, which indicates that messages specified by the *selector* are to be written to all logged-in users.

Blank lines are ignored. Lines for which the first character is a '#' are treated as comments.

Sun386i DESCRIPTION

The file is as described above, except that there is an additional valid entry type, for translation. A line containing the keyword "translate," if present, specifies how system error messages are translated, suppressed, or forwarded to appropriate log files and/or users.

A translation entry in the file is composed of five TAB-separated fields:

translate *source* *facility* *input* *output*

The *translate* field consists of the word **translate** and is used to indicate that this is a translation entry.

The *source* field contains a comma separated list of source names. Recognized sources are:

klog Messages placed in **/dev/klog** by the kernel.

log Messages placed in **/dev/log** file by local programs.

syslog Messages placed in the internet socket by programs on other systems.

* An asterisk indicates all three sources (**klog**, **log** and **syslog**).

The *facility* field contains a comma-separated list of facilities.

The *input* field is the name of the file used to map error messages (in printf format strings) to numbers. This number is used to locate a new string in the file specified in the output field. The format of both files is described in **translate**(5).

The output file specified by the output field translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message.

EXAMPLE

With the following configuration file:

```
*.notice;mail.info        /var/log/notice
*.crit                    /var/log/critical
kern,mark.debug           /dev/console
kern.err                  @server
*.emerg                   *
*.alert                   root,operator
*.alert;auth.warning      /var/log/auth
```

syslogd will log all mail system messages except **debug** messages and all **notice** (or higher) messages into a file named **/var/log/notice**. It logs all critical messages into **/var/log/critical**, and all kernel messages and 20-minute marks onto the system console.

Kernel messages of **err** (error) severity or higher are forwarded to the machine named *server*. Emergency messages are forwarded to all users. The users "root" and "operator" are informed of any **alert** messages. All messages from the authorization system of **warning** level or higher are logged in the file **/var/log/auth**.

FILES

/etc/syslog.conf
/var/log/notice
/var/log/critical
/var/log/auth

SEE ALSO

at(1), **crontab**(1), **logger**(1), **login**(1), **lpr**(1), **m4**(1V), **su**(1V), **syslog**(3), **translate**(5), **cron**(8), **ftpd**(8C), **getty**(8), **lpc**(8), **lpd**(8), **routed**(8C), **syslogd**(8)

NAME

systems – NIS systems file

SYNOPSIS

/etc/systems

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/systems** file is used only by SNAP and Automatic System Installation, and contains basic information about each host on the network. It is an ASCII file in the **/etc** directory on the Network Information Service (NIS) master server. To successfully administer all systems in a NIS domain using SNAP, there must be an entry in this file for each host listed in the **/etc/hosts** file. For each host, this file contains a one-line entry, of the following form, where each field *must* be separated by a TAB character:

system architecture sunos "hostid" memory_size disk_size network_role

system is the name of a host, whether it is on a network or a standalone system. This field contains only lowercase and numeric characters, must start with a lower-case character, and must not be longer than 32 characters.

architecture
indicates the architecture of the specified system. This can be **s386**, **sun4**, **sun3**, **sun2**, **sun1**, **pcnfs**, or **other**.

sunos indicates the SunOS operating system version the system is running. Typically, the form is **sunos***version_number* or **unknown**. SNAP always inserts **unknown** when adding new systems.

hostid the system host ID, as obtained from **/bin/hostid**. This entry must be in quotes. If the host ID is - **unknown**, an empty string (**" "**) is specified. SNAP always inserts an empty string when adding new systems.

memory_size
amount of memory, in kilobytes. This can be **8000** (for 8 megabytes), **4000** (for 4 megabytes), or **–1** for **unknown**. SNAP always inserts **–1** when adding new systems.

disk_size
amount of disk space, in kilobytes. This can be any value, but typically should be close to the actual disk size or to the total amount of disk space, if expansion disks were added. Diskless clients would have a zero value, while **unknown** disk sizes are specified by a **–1** value. SNAP always inserts **–1** when adding new network clients.

network_role
indicates the role the system plays on the network. This can be **master_bootserver**, **slave_bootserver**, **network_client**, or **diskless_client**.

EXAMPLES

Here is a sample systems file:

vulcan	**s386**	**sunos4.0.1**	**"12345678"**	**8000**	**327000**	**master_bootserver**
polaris	**s386**	**sunos4.0.1**	**""**	**8000**	**91000**	**slave_bootserver**
star	**sun4**	**sunos4.1**	**""**	**8000**	**91000**	**network_client**
traveler	**s386**	**sunos4.0.1**	**""**	**8000**	**0**	**diskless_client**

FILES

/etc/systems
/etc/hosts
/bin/hostid

SEE ALSO

snap(1), **vipw**(8)

System and Network Administration,
Sun386i Advanced Administration

NOTES

Take precautions to lock the **/etc/systems** file against simultaneous changes if it will be edited with a text editor; editing with **vipw**(8) provides the necessary locking.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

tar – tape archive file format

DESCRIPTION

tar, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A "tar tape" or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an EOF indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **tar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK512
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
        } dbuf;
};
```

name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width *w*) contains w-2 digits, a SPACE, and a null character, except *size* and *mtime*, which do not contain the trailing null. *name* is the name of the file, as specified on the **tar** command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers which own the file. *size* is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII '0' if the file is "normal" or a special file, ASCII '1' if it is an hard link, and ASCII '2' if it is a symbolic link. The name linked-to, if any, is in *linkname*, with a trailing null character. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given inode number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

SEE ALSO

tar(1)

BUGS

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

term – terminal driving tables for nroff

SYNOPSIS

/usr/lib/term/tab*name*

DESCRIPTION

nroff(1) uses driving tables to customize its output for various types of output devices, such as terminals, line printers, daisy-wheel printers, or special output filter programs. These driving tables are written as C programs, compiled, and installed in the directory **/usr/lib/term**. The *name* of the output device is specified with the **–T** option of **nroff**. The structure of the terminal table is as follows:

```
#define     INCH     240

struct {
            int bset;
            int breset;
            int Hor;
            int Vert;
            int Newline;
            int Char;
            int Em;
            int Halfline;
            int Adj;
            char *twinit;
            char *twrest;
            char *twnl;
            char *hlr;
            char *hlf;
            char *flr;
            char *bdon;
            char *bdoff;
            char *ploton;
            char *plotoff;
            char *up;
            char *down;
            char *right;
            char *left;
            char *codetab[256–32];
            char *zzz;
} t;
```

The meanings of the various fields are as follows:

bset Bits to set in the **sg_flags** field of the **sgtty** structure before output; see **ttcompat**(4M).

breset Bits to reset in the **sg_flags** field of the **sgtty** structure after output; see **ttcompat**(4M).

Hor Horizontal resolution in fractions of an inch.

Vert Vertical resolution in fractions of an inch.

Newline Space moved by a NEWLINE (LINEFEED) character in fractions of an inch.

Char Quantum of character sizes, in fractions of an inch (that is, a character is a multiple of **Char** units wide).

Em Size of an em in fractions of an inch.

Halfline Space moved by a half-LINEFEED (or half-reverse-LINEFEED) character in fractions of an inch.

Adj	Quantum of white space, in fractions of an inch. (that is, white spaces are a multiple of **Adj** units wide) Note: if this is less than the size of the SPACE character (in units of **Char**; see below for how the sizes of characters are defined), **nroff** will output fractional SPACE characters using plot mode. Also, if the **–e** switch to **nroff** is used, **Adj** is set equal to **Hor** by **nroff**.
twinit	Set of characters used to initialize the terminal in a mode suitable for **nroff**.
twrest	Set of characters used to restore the terminal to normal mode.
twnl	Set of characters used to move down one line.
hlr	Set of characters used to move up one-half line.
hlf	Set of characters used to move down one-half line.
flr	Set of characters used to move up one line.
bdon	Set of characters used to turn on hardware boldface mode, if any.
bdoff	Set of characters used to turn off hardware boldface mode, if any.
ploton	Set of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.
plotoff	Set of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.
up	Set of characters used to move up one resolution unit (**Vert**) in plot mode, if any.
down	Set of characters used to move down one resolution unit (**Vert**) in plot mode, if any.
right	Set of characters used to move right one resolution unit (**Hor**) in plot mode, if any.
left	Set of characters used to move left one resolution unit (**Hor**) in plot mode, if any.
codetab	Definition of characters needed to print an **nroff** character on the terminal. The first byte is the number of character units (**Char**) needed to hold the character; that is, **\001** is one unit wide, **\002** is two units wide, etc. The high-order bit (0200) is on if the character is to be underlined in underline mode (**.ul**). The rest of the bytes are the characters used to produce the character in question. If the character has the sign (0200) bit on, it is a code to move the terminal in plot mode. It is encoded as:

0100 bit on	vertical motion.
0100 bit off	horizontal motion.
040 bit on	negative (up or left) motion.
040 bit off	positive (down or right) motion.
037 bits	number of such motions to make.

zzz	A zero terminator at the end.

All quantities which are in units of fractions of an inch should be expressed as '**INCH*_num/denom_**', where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as '**INCH/48**'.

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The following is a sample **codetab** encoding.

```
"\001 ",        /*space*/
"\001!",        /*!*/
"\001\"",       /*"*/
"\001#",        /*#*/
"\001$",        /*$*/
```

```
"\001%",        /*%*/
"\001&",        /*&*/
"\001'",        /*'*/
"\001(",        /*(*/
"\001)",        /*)*/
"\001*",        /***/
"\001+",        /*+*/
"\001,",        /*,*/
"\001-",        /*-*/
"\001.",        /*.*/
"\001/",        /*/*/
"\2010",        /*0*/
"\2011",        /*1*/
"\2012",        /*2*/
"\2013",        /*3*/
"\2014",        /*4*/
"\2015",        /*5*/
"\2016",        /*6*/
"\2017",        /*7*/
"\2018",        /*8*/
"\2019",        /*9*/
"\001:",        /*:*/
"\001;",        /*;*/
"\001<",        /*<*/
"\001=",        /*=*/
"\001>",        /*>*/
"\001?",        /*?*/
"\001@",        /*@*/
"\201A",        /*A*/
"\201B",        /*B*/
"\201C",        /*C*/
"\201D",        /*D*/
"\201E",        /*E*/
"\201F",        /*F*/
"\201G",        /*G*/
"\201H",        /*H*/
"\201I",        /*I*/
"\201J",        /*J*/
"\201K",        /*K*/
"\201L",        /*L*/
"\201M",        /*M*/
"\201N",        /*N*/
"\201O",        /*O*/
"\201P",        /*P*/
"\201Q",        /*Q*/
"\201R",        /*R*/
"\201S",        /*S*/
"\201T",        /*T*/
"\201U",        /*U*/
"\201V",        /*V*/
"\201W",        /*W*/
"\201X",        /*X*/
"\201Y",        /*Y*/
```

```
"\201Z",                /*Z*/
"\001[",                /*[*/
"\001\\",               /*\*/
"\001]",                /*]*/
"\001^",                /*^*/
"\001_",                /*_*/
"\001`",                /*`*/
"\201a",                /*a*/
"\201b",                /*b*/
"\201c",                /*c*/
"\201d",                /*d*/
"\201e",                /*e*/
"\201f",                /*f*/
"\201g",                /*g*/
"\201h",                /*h*/
"\201i",                /*i*/
"\201j",                /*j*/
"\201k",                /*k*/
"\201l",                /*l*/
"\201m",                /*m*/
"\201n",                /*n*/
"\201o",                /*o*/
"\201p",                /*p*/
"\201q",                /*q*/
"\201r",                /*r*/
"\201s",                /*s*/
"\201t",                /*t*/
"\201u",                /*u*/
"\201v",                /*v*/
"\201w",                /*w*/
"\201x",                /*x*/
"\201y",                /*y*/
"\201z",                /*z*/
"\001{",                /*{*/
"\001|",                /*|*/
"\001}",                /*}*/
"\001~",                /*~*/
"\000\0",               /*narrow sp*/
"\001-",                /*hyphen*/
"\001\016Z\017",        /*bullet*/
"\002[]",               /*square*/
"\002--",               /*3/4 em dash*/
"\001_",                /*rule*/
"\0031/4",              /*1/4*/
"\0031/2",              /*1/2*/
"\0033/4",              /*3/4*/
"\001-",                /*minus*/
"\202fi",               /*fi*/
"\202fl",               /*fl*/
"\202ff",               /*ff*/
"\203ffi",              /*ffi*/
"\203ffl",              /*ffl*/
"\001\016p\017",        /*degree*/
```

```
"\001|\b\342-\302",             /*dagger*/
"\001\301s\343s\302",           /*section*/
"\001'",                        /*foot mark*/
"\001\033Z",                    /*acute accent*/
"\001`",                        /*grave accent*/
"\001_",                        /*underrule*/
"\001/",                        /*long slash*/
"\000\0",                       /*half narrow space*/
"\001 ",                        /*unpaddable space*/
"\001\016A\017",                /*alpha*/
"\001\016B\017",                /*beta*/
"\001\016C\017",                /*gamma*/
"\001\016D\017",                /*delta*/
"\001\016E\017",                /*epsilon*/
"\001\016F\017",                /*zeta*/
"\001\016G\017",                /*eta*/
"\001\016H\017",                /*theta*/
"\001\016I\017",                /*iota*/
"\001\016J\017",                /*kappa*/
"\001\016K\017",                /*lambda*/
"\001\016L\017",                /*mu*/
"\001\016M\017",                /*nu*/
"\001\016N\017",                /*xi*/
"\001\016O\017",                /*omicron*/
"\001\016P\017",                /*pi*/
"\001\016Q\017",                /*rho*/
"\001\016R\017",                /*sigma*/
"\001\016S\017",                /*tau*/
"\001\016T\017",                /*upsilon*/
"\001\016U\017",                /*phi*/
"\001\016V\017",                /*chi*/
"\001\016W\017",                /*psi*/
"\001\016X\017",                /*omega*/
"\001\016#\017",                /*Gamma*/
"\001\016$\017",                /*Delta*/
"\001\016(\017",                /*Theta*/
"\001\016+\017",                /*Lambda*/
"\001\016.\017",                /*Xi*/
"\001\0160\017",                /*Pi*/
"\001\0169\017",                /*Sigma*/
"\000",                         /**/
"\001\0164\017",                /*Upsilon*/
"\001\0165\017",                /*Phi*/
"\001\0167\017",                /*Psi*/
"\001\0168\017",                /*Omega*/
"\001\016[\017",                /*square root*/
"\001\016Y\017",                /*\(ts yields script-l*/
"\001\016k\017",                /*root en*/
"\001>\b_",                     /*>=*/
"\001<\b_",                     /*<=*/
"\001=\b_",                     /*identically equal*/
"\001-",                        /*equation minus*/
"\001\016o\017",                /*approx =*/
```

```
"\001\016n\017",                                  /*approximates*/
"\001=\b/",                                       /*not equal*/
"\002-\242-\202>",                                /*right arrow*/
"\002<\b\202-\242\200-",                          /*left arrow*/
"\001|\b^",                                       /*up arrow*/
"\001|\b\302v\342",                               /*down arrow*/
"\001=",                                          /*equation equal*/
"\001\016|\017",                                  /*multiply*/
"\001\016}\017",                                  /*divide*/
"\001\016j\017",                                  /*plus-minus*/
"\001\243|\203_\203|\243",                        /*cup (union)*/
"\001\243|\203\351_\311\203|\243",                /*cap (intersection)*/
"\001\243(\203\302-\345-\303",                    /*subset of*/
"\001\302-\345-\303\203)\243",                    /*superset of*/
"\001_\b\243(\203\302-\345-\303",                 /*improper subset*/
"\001_\b\302-\345-\303\203)\243",                 /*improper superset*/
"\001\016~\017",                                  /*infinity*/
"\001\200o\201\301`\241\341`\241\341`\201\301",   /*partial derivative*/
"\001\016:\017",                                  /*gradient*/
"\001\200-\202\341,\301\242",                     /*not*/
"\001\016?\017",                                  /*integral sign*/
"\002o\242c\202",                                 /*proportional to*/
"\001O\b/",                                       /*empty set*/
"\001<\b\341-\302",                               /*member of*/
"\001+",                                          /*equation plus*/
"\003(R)",                                        /*registered*/
"\003(C)",                                        /*copyright*/
"\001|",                                          /*box rule */
"\001\033Y",                                      /*cent sign*/
"\001|\b\342=\302",                               /*double dagger*/
"\002=>",                                         /*right hand*/
"\002<=",                                         /*left hand*/
"\001*",                                          /*math * */
"\001\0162\017",                                  /*\(bs yields small sigma*/
"\001|",                                          /*or (was star)*/
"\001O",                                          /*circle*/
"\001|",                                          /*left top of big brace*/
"\001|",                                          /*left bot of big brace*/
"\001|",                                          /*right top of big brace*/
"\001|",                                          /*right bot of big brace*/
"\001\016]\017",                                  /*left center of big brace*/
"\001\016\\\017",                                 /*right center of big brace*/
"\001|",                                          /*bold vertical*/
"\001|",                                          /*left floor (lb of big bracket)*/
"\001|",                                          /*right floor (rb of big bracket)*/
"\001|",                                          /*left ceiling (lt of big bracket)*/
"\001|"                                           /*right ceiling (rt of big bracket)*/
```

FILES

/usr/lib/term/tab*name*	driving tables
/usr/lib/term/README	list of terminals supported by **nroff**(1)

SEE ALSO

nroff(1), **ttcompat**(4M)

NAME

term – format of compiled term file

SYNOPSIS

term

DESCRIPTION

Compiled **terminfo** descriptions are placed under the directory **/usr/share/lib/terminfo.** In order to avoid a linear search of a huge system directory, a two-level scheme is used: **/usr/share/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/share/lib/terminfo/a/act4.** Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the **tic**(8V) program, and read by the routine **setupterm** (see **curses**(3V)). Both of these pieces of software are part of **curses**(3V). The file is divided into six parts:

the header,
terminal names,
boolean flags,
numbers,
strings,
and
string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are:

(1) the magic number (octal 0432);
(2) the size, in bytes, of the names section;
(3) the number of bytes in the boolean section;
(4) the number of short integers in the numbers section;
(5) the number of offsets (short integers) in the strings section;
(6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value −1 is represented by 0377, 0377, other negative value are illegal. The −1 generally means that a capability is missing from this terminal. Note: this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is −1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of −1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in **^X** or **\c** notation are stored in their interpreted form, not the printing representation. Padding information **$<*nn*>** and parameter information *%x* are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null-terminated.

Note: it is possible for **setupterm** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm** has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm** must be prepared for both possibilities — this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
cr=^M, cudl=^J, ind=^J, bel=^G, am, cubl=^H,
ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
cols#80, lines#24, cufl=^X, cuul=^Z, home=^],
```

```
000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020   o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040   t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100  \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120 377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140  \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160 377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/share/lib/terminfo/*/*
compiled terminal capability data base

SEE ALSO

curses(3V), **terminfo**(5V), **tic**(8V)

NAME

termcap – terminal capability data base

DESCRIPTION

termcap is a data base describing the capabilities of terminals. Terminals are described in **termcap** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs such as **vi**(1), and libraries such as **curses**(3V), so they can work with a variety of terminals without changes to the programs.

Each **termcap** entry consist of a number of colon-separated (**:**) fields. The first field for each terminal lists the various names by which it is known, separated by bar (|) characters. The first name is always two characters long, and is used by older (version 6) systems (which store the terminal type in a 16-bit word in a system-wide database). The second name given is the most common abbreviation for the terminal (this is the one to which the environment variable **TERM** would normally be set). The last name should fully identify the terminal's make and model. All other names are taken as synonyms for the initial terminal name. All names but the first and last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for added readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

Suffix	*Meaning*	*Example*
–w	wide mode (more than 80 columns)	**vt100–w**
–am	with automatic margins (usually default)	**vt100–am**
–nam	without automatic margins	**vt100–nam**
–*n*	number of lines on the screen	**aaa–60**
–na	no arrow keys (leave them in local)	**concept100–na**
–*n***p**	number of pages of memory	**concept100–4p**
–rv	reverse video	**concept100–rv**

Terminal entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with **#**.

Types of Capabilities

Terminal capabilities each have a two-letter code, and are of three types:

boolean These indicate particular features of the terminal. For instance, an entry for a terminal that has automatic margins (an automatic RETURN and LINEFEED when the end of a line is reached) would contain a field with the boolean capability **am**.

numeric These give the size of the display of some other attribute. Numeric capabilities are followed by the character '**#**', and a number. An entry for a teminal with an 80-column display would have a field containing **co#80**.

string These indicate the character sequences used to perform particular terminal operations. String-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, followed by the character '=', and a string (which ends at the following **:** field delimiter).

A delay factor, in milliseconds may appear after the '='. Padding characters are supplied by **tputs** after the remainder of the string is sent. The delay can be either a number, or a number followed by the character '*', which indicates that the proportional padding is required, in which case the number given is the amount of padding for each line affected by an operation using that capability. (In the case of an insert-character operation, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.)

When a * is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

Comments

To comment-out a capability field, insert a '**.**' (period) as the first character in that field (following the **:**).

Escape Sequence Codes

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

\E	maps to ESC
^X	maps to CTRL-*X* for any appropriate character *X*
\n	maps to LINEFEED
\r	maps to RETURN
\t	maps to TAB
\b	maps to BACKSPACE
\f	maps to FORMFEED

Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret) and \ (backslash) may be given as \^ and \\ respectively.

If it is necessary to place a **:** in a capability it must be escaped in octal as **\072**.

If it is necessary to place a NUL character in a string capability it must be encoded as **\200**. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a **\200** comes out as a **\000** would.)

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3V)-like escapes (*%x*) in it; other characters are passed through unchanged. For example, to address the cursor, the **cm** capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous **CM** capability.)

The % escapes have the following meanings:

%%	produce the character *%*
%d	output *value* as in **printf %d**
%2	output *value* as in **printf %2d**
%3	output *value* as in **printf %3d**
%.	output *value* as in **printf %c**
%+x	add *x* to *value*, then do '*%.*'
%>xy	if *value* > *x* then add *y*, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16*(*value*/10)) + (*value*%10), no output
%D	Reverse coding (*value* – 2*(*value*%16)), no output (Delta Data)

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note: the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is '**:cm=6\E&%r%2c%2Y:**'. Terminals that use '**%.**' need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that TAB characters are not expanded, making **\t** safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM–3a, which offsets row and column by a blank character, thus it requires '**:cm=\E=%+ %+:**'.

Row or column absolute cursor addressing can be given as single-parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions (for example, move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the **tset (1)** program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will set the appropriate delay bits in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD **tset**, the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last*, and the combined length of the entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where *xx* is the capability. For example, the entry

hn|2621–nl:ks@:ke@:tc=2621:

defines a **2621–nl** that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

CAPABILITIES

The characters in the *Notes* field in the next table have the following meanings (more than one may apply to a capability):

- **N** indicates numeric parameter(s)
- **P** indicates that padding may be specified
- ***** indicates that padding may be based on the number of lines affected
- **o** indicates capability is obsolete

Obsolete capabilities have no **terminfo** equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them.

Name	*Type*	*Notes*	*Description*
!1	*str*		sent by shifted save key
!2	*str*		sent by shifted suspend key
!3	*str*		sent by shifted undo key
#1	*str*		sent by shifted help key
#2	*str*		sent by shifted home key
#3	*str*		sent by shifted input key
#4	*str*		sent by shifted left-arrow key
%0	*str*		sent by redo key
%1	*str*		sent by help key

%2	*str*		sent by mark key
%3	*str*		sent by message key
%4	*str*		sent by move key
%5	*str*		sent by next-object key
%6	*str*		sent by open key
%7	*str*		sent by options key
%8	*str*		sent by previous-object key
%9	*str*		sent by print or copy key
%a	*str*		sent by shifted message key
%b	*str*		sent by shifted move key
%c	*str*		sent by shifted next-object key
%d	*str*		sent by shifted options key
%e	*str*		sent by shifted previous-object key
%f	*str*		sent by shifted print or copy key
%g	*str*		sent by shifted redo key
%h	*str*		sent by shifted replace key
%i	*str*		sent by shifted right-arrow key
%j	*str*		sent by shifted resume key
&0	*str*		sent by shifted cancel key
&1	*str*		sent by ref(erence) key
&2	*str*		sent by refresh key
&3	*str*		sent by replace key
&4	*str*		sent by restart key
&5	*str*		sent by resume key
&6	*str*		sent by save key
&7	*str*		sent by suspend key
&8	*str*		sent by undo key
&9	*str*		sent by shifted beg(inning) key
***0**	*str*		sent by shifted find key
***1**	*str*		sent by shifted cmd (command) key
***2**	*str*		sent by shifted copy key
***3**	*str*		sent by shifted create key
***4**	*str*		sent by shifted delete-char key
***5**	*str*		sent by shifted delete-line key
***6**	*str*		sent by select key
***7**	*str*		sent by shifted end key
***8**	*str*		sent by shifted clear-line key
***9**	*str*		sent by shifted exit key
5i	*bool*		printer will not echo on screen
@0	*str*		sent by find key
@1	*str*		sent by beg(inning) key
@2	*str*		sent by cancel key
@3	*str*		sent by close key
@4	*str*		sent by cmd (command) key
@5	*str*		sent by copy key
@6	*str*		sent by create key
@7	*str*		sent by end key
@8	*str*		sent by enter/send key (unreliable)
@9	*str*		sent by exit key
AL	*str*	*(NP*)*	add *n* new blank lines
CC	*str*		terminal settable command character in prototype
CM	*str*	*(NP)*	memory-relative cursor motion to row *m*, column *n*
DC	*str*	*(NP*)*	delete *n* characters
DL	*str*	*(NP*)*	delete *n* lines
DO	*str*	*(NP*)*	move cursor down *n* lines
EP	*bool*	*(o)*	even parity
F1-F9	*str*		sent by function keys 11-19
FA-FZ	*str*		sent by function keys 20-45

Fa-Fr	*str*		sent by function keys 46-63
HC	*bool*		cursor is hard to see
HD	*bool*	*(o)*	half-duplex
IC	*str*	*(NP*)*	insert *n* blank characters
K1	*str*		sent by keypad upper left
K2	*str*		sent by keypad center
K3	*str*		sent by keypad upper right
K4	*str*		sent by keypad lower left
K5	*str*		sent by keypad lower right
LC	*bool*	*(o)*	lower-case only
LE	*str*	*(NP)*	move cursor left *n* positions
LF	*str*	*(P)*	turn off soft labels
LO	*str*	*(P)*	turn on soft labels
MC	*str*	*(P)*	clear left and right soft margins
ML	*str*	*(P)*	set soft left margin
MR	*str*	*(P)*	set soft right margin
NL	*bool*	*(o)*	**\n** is NEWLINE, not LINEFEED
NP	*bool*		pad character does not exist
NR	*bool*		**ti** does not reverse **te**
Nl	*num*		number of labels on screen (start at 1)
OP	*bool*	*(o)*	odd parity
RA	*str*	*(P)*	turn off automatic margins
RF	*str*		send next input character (for ptys)
RI	*str*	*(NP)*	move cursor right *n* positions
RX	*str*	*(P)*	turn off xoff/xon handshaking
SA	*str*	*(P)*	turn on automatic margins
SF	*str*	*(NP*)*	scroll forward *n* lines
SR	*str*	*(NP*)*	scroll backward *n* lines
SX	*str*	*(P)*	turn on xoff/xon handshaking
UC	*bool*	*(o)*	upper-case only
UP	*str*	*(NP*)*	move cursor up *n* lines
XF	*str*		x-off character (default DC3)
XN	*str*		x-on character (default DC1)
ac	*str*		graphic character set pairs aAbBcC – def=VT100
ae	*str*	*(P)*	end alternate character set
al	*str*	*(P*)*	add new blank line
am	*bool*		terminal has automatic margins
as	*str*	*(P)*	start alternate character set
bc	*str*	*(o)*	backspace if not **^H**
bl	*str*	*(P)*	audible signal (bell)
bs	*bool*	*(o)*	terminal can backspace with **^H**
bt	*str*	*(P)*	back-tab
bw	*bool*		**le** (backspace) wraps from column 0 to last column
cb	*str*	*(P)*	clear to beginning of line, inclusive
cd	*str*	*(P*)*	clear to end of display
ce	*str*	*(P)*	clear to end of line
ch	*str*	*(NP)*	set cursor column (horizontal position)
cl	*str*	*(P*)*	clear screen and home cursor
cm	*str*	*(NP)*	screen-relative cursor motion to row *m*, column *n*
co	*num*		number of columns in a line
cr	*str*	*(P*)*	RETURN
cs	*str*	*(NP)*	change scrolling region to lines *m* through *n* (VT100)
ct	*str*	*(P)*	clear all tab stops
cv	*str*	*(NP)*	set cursor row (vertical position)
dB	*num*	*(o)*	milliseconds of **bs** delay needed (default 0)
dC	*num*	*(o)*	milliseconds of **cr** delay needed (default 0)
dF	*num*	*(o)*	milliseconds of **ff** delay needed (default 0)
dN	*num*	*(o)*	milliseconds of **nl** delay needed (default 0)

dT	*num*	*(o)*	milliseconds of horizontal tab delay needed (default 0)
dV	*num*	*(o)*	milliseconds of vertical tab delay needed (default 0)
da	*bool*		display may be retained above the screen
db	*bool*		display may be retained below the screen
dc	*str*	*(P*)*	delete character
dl	*str*	*(P*)*	delete line
dm	*str*		enter delete mode
do	*str*		down one line
ds	*str*		disable status line
eA	*str*	*(P)*	enable graphic character set
ec	*str*	*(NP)*	erase *n* characters
ed	*str*		end delete mode
ei	*str*		end insert mode
eo	*bool*		can erase overstrikes with a blank
es	*bool*		escape can be used on the status line
ff	*str*	*(P*)*	hardcopy terminal page eject
fs	*str*		return from status line
gn	*bool*		generic line type (for example dialup, switch)
hc	*bool*		hardcopy terminal
hd	*str*		half-line down (forward 1/2 linefeed)
ho	*str*	*(P)*	home cursor
hs	*bool*		has extra "status line"
hu	*str*		half-line up (reverse 1/2 linefeed)
hz	*bool*		cannot print ~s (Hazeltine)
i1	*str*		terminal initialization string (**terminfo** only)
i3	*str*		terminal initialization string (**terminfo** only)
iP	*str*		pathname of program for initialization (**terminfo** only)
ic	*str*	*(P*)*	insert character
if	*str*		name of file containing initialization string
im	*str*		enter insert mode
in	*bool*		insert mode distinguishes nulls
ip	*str*	*(P*)*	insert pad after character inserted
is	*str*		terminal initialization string
it	*num*		tab stops initially every *n* positions
k0-k9	*str*		sent by function keys 0-9
k;	*str*		sent by function key 10
kA	*str*		sent by insert-line key
kB	*str*		sent by back-tab key
kC	*str*		sent by clear-screen or erase key
kD	*str*		sent by delete-character key
kE	*str*		sent by clear-to-end-of-line key
kF	*str*		sent by scroll-forward/down key
kH	*str*		sent by home-down key
kI	*str*		sent by insert-character or enter-insert-mode key
kL	*str*		sent by delete-line key
kM	*str*		sent by insert key while in insert mode
kN	*str*		sent by next-page key
kP	*str*		sent by previous-page key
kR	*str*		sent by scroll-backward/up key
kS	*str*		sent by clear-to-end-of-screen key
kT	*str*		sent by set-tab key
ka	*str*		sent by clear-all-tabs key
kb	*str*		sent by backspace key
kd	*str*		sent by down-arrow key
ke	*str*		out of "keypad transmit" mode
kh	*str*		sent by home key
kl	*str*		sent by left-arrow key
km	*bool*		has a "meta" key (shift, sets parity bit)

kn	*num*	*(o)*	number of function (**k0–k9**) keys (default 0)
ko	*str*	*(o)*	termcap entries for other non-function keys
kr	*str*		sent by right-arrow key
ks	*str*		put terminal in "keypad transmit" mode
kt	*str*		sent by clear-tab key
ku	*str*		sent by up-arrow key
l0-l9	*str*		labels on function keys 0-9 if not f0-f9
la	*str*		label on function key 10 if not f10
le	*str*	*(P)*	move cursor left one position
lh	*num*		number of rows in each label
li	*num*		number of lines on screen or page
ll	*str*		last line, first column
lm	*num*		lines of memory if > **li** (0 means varies)
lw	*num*		number of columns in each label
ma	*str*	*(o)*	arrow key map (used by *vi* version 2 only)
mb	*str*		turn on blinking attribute
md	*str*		turn on bold (extra bright) attribute
me	*str*		turn off all attributes
mh	*str*		turn on half-bright attribute
mi	*bool*		safe to move while in insert mode
mk	*str*		turn on blank attribute (characters invisible)
ml	*str*	*(o)*	memory lock on above cursor
mm	*str*		turn on "meta mode" (8th bit)
mo	*str*		turn off "meta mode"
mp	*str*		turn on protected attribute
mr	*str*		turn on reverse-video attribute
ms	*bool*		safe to move in standout modes
mu	*str*	*(o)*	memory unlock (turn off memory lock)
nc	*bool*	*(o)*	no correctly-working **cr** (Datamedia 2500, Hazeltine 2000)
nd	*str*		non-destructive space (cursor right)
nl	*str*	*(o)*	NEWLINE character if not
ns	*bool*	*(o)*	terminal is a CRT but does not scroll
nw	*str*	*(P)*	NEWLINE (behaves like **cr** followed by **do**)
nx	*bool*		padding will not work, xoff/xon required
os	*bool*		terminal overstrikes
pO	*str*	*(N)*	turn on the printer for *n* bytes
pb	*num*		lowest baud where delays are required
pc	*str*		pad character (default NUL)
pf	*str*		turn off the printer
pk	*str*		program function key *n* to type string *s* (**terminfo** only)
pl	*str*		program function key *n* to execute string *s* (**terminfo** only)
pn	*str*	*(NP)*	program label *n* to show string *s* (**terminfo** only)
po	*str*		turn on the printer
ps	*str*		print contents of the screen
pt	*bool*	*(o)*	has hardware tab stops (may need to be set with **is**)
px	*str*		program function key *n* to transmit string *s* (**terminfo** only)
r1	*str*		reset terminal completely to sane modes (**terminfo** only)
r2	*str*		reset terminal completely to sane modes (**terminfo** only)
r3	*str*		reset terminal completely to sane modes (**terminfo** only)
rP	*str*	*(P)*	like **ip** but when in replace mode
rc	*str*	*(P)*	restore cursor to position of last **sc**
rf	*str*		name of file containing reset string
ri	*?*		unknown at present
rp	*str*	*(NP*)*	repeat character *c n* times
rs	*str*		reset terminal completely to sane modes
sa	*str*	*(NP)*	define the video attributes (9 parameters)
sc	*str*	*(P)*	save cursor position
se	*str*		end standout mode

sf	*str*	*(P)*	scroll text up
sg	*num*		number of garbage chars left by **so** or **se** (default 0)
so	*str*		begin standout mode
sr	*str*	*(P)*	scroll text down
st	*str*		set a tab stop in all rows, current column
ta	*str*	*(P)*	move cursor to next 8-position hardware tab stop
tc	*str*		entry of similar terminal – must be last
te	*str*		string to end programs that use **termcap**
ti	*str*		string to begin programs that use **termcap**
ts	*str*	*(N)*	go to status line, column *n*
uc	*str*		underscore one character and move past it
ue	*str*		end underscore mode
ug	*num*		number of garbage chars left by **us** or **ue** (default 0)
ul	*bool*		underline character overstrikes
up	*str*		upline (cursor up)
us	*str*		start underscore mode
vb	*str*		visible bell (must not move cursor)
ve	*str*		make cursor appear normal (undo **vs**/**vi**)
vi	*str*		make cursor invisible
vs	*str*		make cursor very visible
vt	*num*		virtual terminal number (not supported on all systems)
wi	*str*	*(N)*	set current window to lines *i* through *j*, columns *m* through *n*
ws	*num*		number of columns in status line
xb	*bool*		Beehive (f1=ESC, f2=^C)
xn	*bool*		NEWLINE ignored after 80 cols (Concept)
xo	*bool*		terminal uses xoff/xon handshaking
xr	*bool*	*(o)*	RETURN acts like **ce cr nl** (Delta Data)
xs	*bool*		standout not erased by overwriting (Hewlett-Packard)
xt	*bool*		TAB characters destructive, magic **so** char (Teleray 1061)
xx	*bool*	*(o)*	Tektronix 4025 insert-line

ENVIRONMENT

If the environment variable **TERMCAP** contains an absolute pathname, programs look to that file for terminal descriptions, rather than **/usr/share/lib/termcap**. If the value of this variable is in the form of a **termcap** entry, programs use that value for the terminal description.

FILES

/usr/share/lib/termcap file containing terminal descriptions

SEE ALSO

ex(1), **more**(1), **tset**(1), **ul**(1), **vi**(1), **curses**(3V), **printf**(3V), **termcap**(3X), **term**(5V), **terminfo**(5V)

System and Network Administration

WARNINGS

UNIX System V uses **terminfo**(5V) rather than **termcap**. SunOS supports either **termcap** or **terminfo**(5V) terminal databases, depending on whether you link with the **termcap**(3X) or **curses**(3V) libraries. Transitions between the two should be relatively painless if capabilities flagged as "obsolete" are avoided.

vi allows only 256 characters for string capabilities, and the routines in **termcap**(3X) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped NEWLINE characters) may not exceed 1024.

Not all programs support all entries.

NAME

terminfo – terminal capability data base

SYNOPSIS

/usr/share/lib/terminfo/?/*

AVAILABILITY

This database is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

terminfo is a compiled database (see **tic**(8V)) describing the capabilities of terminals. Terminals are described in **terminfo** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, and by libraries such as **curses**(3V), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **–I** option of **infocmp**(8V).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the **terminfo** database gives the name by which **terminfo** knows the terminal, separated by pipe (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to which the environment variable **TERM** would normally be set), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

Suffix	*Meaning*	*Example*
–w	wide mode (more than 80 columns)	**vt100–w**
–am	with automatic margins (usually default)	**vt100–am**
–nam	without automatic margins	**vt100–nam**
–*n*	number of lines on the screen	**aaa–60**
–na	no arrow keys (leave them in local)	**concept100–na**
–*n*p	number of pages of memory	**concept100–4p**
–rv	reverse video	**concept100–rv**

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the **terminfo** level) accesses the capability. The **capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the **tput**(1V) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that corresponds to the old **termcap** capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the table below, have names beginning with '**key_**'. The following indicators may appear at the end of the **Description** for a variable.

(G) indicates that the string is passed through **tparm**( ) with parameters (parms) as given ($\#_i$).
(*) indicates that padding may be based on the number of lines affected.
($\#_i$) indicates the i^{th} parameter.

Variable	*Capname*	*Termcap*	*Description*
Boolean			
auto_left_margin	**bw**	**bw**	**cub**1 wraps from column 0 to last column
auto_right_margin	**am**	**am**	Terminal has automatic margins
no_esc_ctlc	**xsb**	**xb**	Beehive (f1=ESC, f2=^C)
ceol_standout_glitch	**xhp**	**xs**	Standout not erased by overwriting (Hewlett-Packard)
eat_newline_glitch	**xenl**	**xn**	NEWLINE ignored after 80 cols (Concept)
erase_overstrike	**eo**	**eo**	Can erase overstrikes with a blank
generic_type	**gn**	**gn**	Generic line type (for example, dialup, switch).
hard_copy	**hc**	**hc**	Hardcopy terminal
hard_cursor	**chts**	**HC**	Cursor is hard to see
has_meta_key	**km**	**km**	Has a meta key (shift, sets parity bit)
has_status_line	**hs**	**hs**	Has extra "status line"
insert_null_glitch	**in**	**in**	Insert mode distinguishes nulls
memory_above	**da**	**da**	Display may be retained above the screen
memory_below	**db**	**db**	Display may be retained below the screen
move_insert_mode	**mir**	**mi**	Safe to move while in insert mode
move_standout_mode	**msgr**	**ms**	Safe to move in standout modes
needs_xon_xoff	**nxon**	**nx**	Padding will not work, xon/xoff required
non_rev_rmcup	**nrrmc**	**NR**	**smcup** does not reverse **rmcup**
no_pad_char	**npc**	**NP**	Pad character does not exist
over_strike	**os**	**os**	Terminal overstrikes on hard-copy terminal
prtr_silent	**mc5i**	**5i**	Printer will not echo on screen
status_line_esc_ok	**eslok**	**es**	Escape can be used on the status line
dest_tabs_magic_smso	**xt**	**xt**	Destructive TAB characters, magic **smso** char (Teleray 1061)
tilde_glitch	**hz**	**hz**	Hazeltine; cannot print tildes(~)
transparent_underline	**ul**	**ul**	Underline character overstrikes
xon_xoff	**xon**	**xo**	Terminal uses xon/xoff handshaking
Number			
columns	**cols**	**co**	Number of columns in a line
init_tabs	**it**	**it**	tab stops initially every # spaces
label_height	**lh**	**lh**	Number of rows in each label
label_width	**lw**	**lw**	Number of cols in each label
lines	**lines**	**li**	Number of lines on screen or page
lines_of_memory	**lm**	**lm**	Lines of memory if > **lines**; **0** means varies
magic_cookie_glitch	**xmc**	**sg**	Number blank chars left by **smso** or **rmso**
num_labels	**nlab**	**Nl**	Number of labels on screen (start at 1)
padding_baud_rate	**pb**	**pb**	Lowest baud rate where padding needed
virtual_terminal	**vt**	**vt**	Virtual terminal number (not supported on all systems)
width_status_line	**wsl**	**ws**	Number of columns in status line
String			
acs_chars	**acsc**	**ac**	Graphic charset pairs aAbBcC - def=VT100
back_tab	**cbt**	**bt**	Back tab
bell	**bel**	**bl**	Audible signal (bell)
carriage_return	**cr**	**cr**	RETURN (*)
change_scroll_region	**csr**	**cs**	Change to lines #1 through #2 (VT100) (G)

char_padding	**rmp**	**rP**	Like **ip** but when in replace mode
clear_all_tabs	**tbc**	**ct**	Clear all tab stops
clear_margins	**mgc**	**MC**	Clear left and right soft margins
clear_screen	**clear**	**cl**	Clear screen and home cursor (*)
clr_bol	**el1**	**cb**	Clear to beginning of line, inclusive
clr_eol	**el**	**ce**	Clear to end of line
clr_eos	**ed**	**cd**	Clear to end of display (*)
column_address	**hpa**	**ch**	Horizontal position absolute (G)
command_character	**cmdch**	**CC**	Terminal settable command char in prototype
cursor_address	**cup**	**cm**	Cursor motion to row #1 col #2 (G)
cursor_down	**cud1**	**do**	Down one line
cursor_home	**home**	**ho**	Home cursor (if no **cup**)
cursor_invisible	**civis**	**vi**	Make cursor invisible
cursor_left	**cub1**	**le**	Move cursor left one SPACE
cursor_mem_address	**mrcup**	**CM**	Memory relative cursor addressing (G)
cursor_normal	**cnorm**	**ve**	Make cursor appear normal (undo **cvvis/civis**)
cursor_right	**cuf1**	**nd**	Non-destructive space (cursor right)
cursor_to_ll	**ll**	**ll**	Last line, first column (if no **cup**)
cursor_up	**cuu1**	**up**	Upline (cursor up)
cursor_visible	**cvvis**	**vs**	Make cursor very visible
delete_character	**dch1**	**dc**	Delete character (*)
delete_line	**dl1**	**dl**	Delete line (*)
dis_status_line	**dsl**	**ds**	Disable status line
down_half_line	**hd**	**hd**	Half-line down (forward 1/2 LINEFEED)
ena_acs	**enacs**	**eA**	Enable alternate char set
enter_alt_charset_mode	**smacs**	**as**	Start alternate character set
enter_am_mode	**smam**	**SA**	Turn on automatic margins
enter_blink_mode	**blink**	**mb**	Turn on blinking
enter_bold_mode	**bold**	**md**	Turn on bold (extra bright) mode
enter_ca_mode	**smcup**	**ti**	String to begin programs that use **cup**
enter_delete_mode	**smdc**	**dm**	Delete mode (enter)
enter_dim_mode	**dim**	**mh**	Turn on half-bright mode
enter_insert_mode	**smir**	**im**	Insert mode (enter);
enter_protected_mode	**prot**	**mp**	Turn on protected mode
enter_reverse_mode	**rev**	**mr**	Turn on reverse video mode
enter_secure_mode	**invis**	**mk**	Turn on blank mode (chars invisible)
enter_standout_mode	**smso**	**so**	Begin standout mode
enter_underline_mode	**smul**	**us**	Start underscore mode
enter_xon_mode	**smxon**	**SX**	Turn on xon/xoff handshaking
erase_chars	**ech**	**ec**	Erase #1 characters (G)
exit_alt_charset_mode	**rmacs**	**ae**	End alternate character set
exit_am_mode	**rmam**	**RA**	Turn off automatic margins
exit_attribute_mode	**sgr0**	**me**	Turn off all attributes
exit_ca_mode	**rmcup**	**te**	String to end programs that use **cup**
exit_delete_mode	**rmdc**	**ed**	End delete mode
exit_insert_mode	**rmir**	**ei**	End insert mode;
exit_standout_mode	**rmso**	**se**	End standout mode
exit_underline_mode	**rmul**	**ue**	End underscore mode
exit_xon_mode	**rmxon**	**RX**	Turn off xon/xoff handshaking
flash_screen	**flash**	**vb**	Visible bell (must not move cursor)
form_feed	**ff**	**ff**	Hardcopy terminal page eject (*)
from_status_line	**fsl**	**fs**	Return from status line
init_1string	**is1**	**i1**	Terminal initialization string
init_2string	**is2**	**is**	Terminal initialization string
init_3string	**is3**	**i3**	Terminal initialization string
init_file	**if**	**if**	Name of initialization file containing **is**
init_prog	**iprog**	**iP**	Path name of program for init
insert_character	**ich1**	**ic**	Insert character

insert_line	**il1**	**al**	Add new blank line (*)
insert_padding	**ip**	**ip**	Insert pad after character inserted (*)
key_a1	**ka1**	**K1**	KEY_A1, 0534, Upper left of keypad
key_a3	**ka3**	**K3**	KEY_A3, 0535, Upper right of keypad
key_b2	**kb2**	**K2**	KEY_B2, 0536, Center of keypad
key_backspace	**kbs**	**kb**	KEY_BACKSPACE, 0407, Sent by BACKSPACE key
key_beg	**kbeg**	**@1**	KEY_BEG, 0542, Sent by beg(inning) key
key_btab	**kcbt**	**kB**	KEY_BTAB, 0541, Sent by back-tab key
key_c1	**kc1**	**K4**	KEY_C1, 0537, Lower left of keypad
key_c3	**kc3**	**K5**	KEY_C3, 0540, Lower right of keypad
key_cancel	**kcan**	**@2**	KEY_CANCEL, 0543, Sent by cancel key
key_catab	**ktbc**	**ka**	KEY_CATAB, 0526, Sent by clear-all-tabs key
key_clear	**kclr**	**kC**	KEY_CLEAR, 0515, Sent by clear- screen or erase key
key_close	**kclo**	**@3**	KEY_CLOSE, 0544, Sent by close key
key_command	**kcmd**	**@4**	KEY_COMMAND, 0545, Sent by cmd (command) key
key_copy	**kcpy**	**@5**	KEY_COPY, 0546, Sent by copy key
key_create	**kcrt**	**@6**	KEY_CREATE, 0547, Sent by create key
key_ctab	**kctab**	**kt**	KEY_CTAB, 0525, Sent by clear-tab key
key_dc	**kdch1**	**kD**	KEY_DC, 0512, Sent by delete-character key
key_dl	**kdl1**	**kL**	KEY_DL, 0510, Sent by delete-line key
key_down	**kcud1**	**kd**	KEY_DOWN, 0402, Sent by terminal down-arrow key
key_eic	**krmir**	**kM**	KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode
key_end	**kend**	**@7**	KEY_END, 0550, Sent by end key
key_enter	**kent**	**@8**	KEY_ENTER, 0527, Sent by enter/send key
key_eol	**kel**	**kE**	KEY_EOL, 0517, Sent by clear-to-end- of-line key
key_eos	**ked**	**kS**	KEY_EOS, 0516, Sent by clear-to-end- of-screen key
key_exit	**kext**	**@9**	KEY_EXIT, 0551, Sent by exit key
key_f0	**kf0**	**k0**	KEY_F(0), 0410, Sent by function key f0
key_f1	**kf1**	**k1**	KEY_F(1), 0411, Sent by function key f1
key_f2	**kf2**	**k2**	KEY_F(2), 0412, Sent by function key f2
key_f3	**kf3**	**k3**	KEY_F(3), 0413, Sent by function key f3
key_f4	**kf4**	**k4**	KEY_F(4), 0414, Sent by function key f4
key_f5	**kf5**	**k5**	KEY_F(5), 0415, Sent by function key f5
key_f6	**kf6**	**k6**	KEY_F(6), 0416, Sent by function key f6
key_f7	**kf7**	**k7**	KEY_F(7), 0417, Sent by function key f7
key_f8	**kf8**	**k8**	KEY_F(8), 0420, Sent by function key f8
key_f9	**kf9**	**k9**	KEY_F(9), 0421, Sent by function key f9
key_f10	**kf10**	**k;**	KEY_F(10), 0422, Sent by function key f10
key_f11	**kf11**	**F1**	KEY_F(11), 0423, Sent by function key f11
key_f12	**kf12**	**F2**	KEY_F(12), 0424, Sent by function key f12
key_f13	**kf13**	**F3**	KEY_F(13), 0425, Sent by function key f13
key_f14	**kf14**	**F4**	KEY_F(14), 0426, Sent by function key f14
key_f15	**kf15**	**F5**	KEY_F(15), 0427, Sent by function key f15
key_f16	**kf16**	**F6**	KEY_F(16), 0430, Sent by function key f16
key_f17	**kf17**	**F7**	KEY_F(17), 0431, Sent by function key f17
key_f18	**kf18**	**F8**	KEY_F(18), 0432, Sent by function key f18
key_f19	**kf19**	**F9**	KEY_F(19), 0433, Sent by function key f19
key_f20	**kf20**	**FA**	KEY_F(20), 0434, Sent by function key f20
key_f21	**kf21**	**FB**	KEY_F(21), 0435, Sent by function key f21
key_f22	**kf22**	**FC**	KEY_F(22), 0436, Sent by function key f22
key_f23	**kf23**	**FD**	KEY_F(23), 0437, Sent by function key f23
key_f24	**kf24**	**FE**	KEY_F(24), 0440, Sent by function key f24
key_f25	**kf25**	**FF**	KEY_F(25), 0441, Sent by function key f25
key_f26	**kf26**	**FG**	KEY_F(26), 0442, Sent by function key f26
key_f27	**kf27**	**FH**	KEY_F(27), 0443, Sent by function key f27
key_f28	**kf28**	**FI**	KEY_F(28), 0444, Sent by function key f28
key_f29	**kf29**	**FJ**	KEY_F(29), 0445, Sent by function key f29
key_f30	**kf30**	**FK**	KEY_F(30), 0446, Sent by function key f30

key_f31	**kf31**	**FL**	KEY_F(31), 0447, Sent by function key f31
key_f32	**kf32**	**FM**	KEY_F(32), 0450, Sent by function key f32
key_f33	**kf33**	**FN**	KEY_F(13), 0451, Sent by function key f13
key_f34	**kf34**	**FO**	KEY_F(34), 0452, Sent by function key f34
key_f35	**kf35**	**FP**	KEY_F(35), 0453, Sent by function key f35
key_f36	**kf36**	**FQ**	KEY_F(36), 0454, Sent by function key f36
key_f37	**kf37**	**FR**	KEY_F(37), 0455, Sent by function key f37
key_f38	**kf38**	**FS**	KEY_F(38), 0456, Sent by function key f38
key_f39	**kf39**	**FT**	KEY_F(39), 0457, Sent by function key f39
key_f40	**kf40**	**FU**	KEY_F(40), 0460, Sent by function key f40
key_f41	**kf41**	**FV**	KEY_F(41), 0461, Sent by function key f41
key_f42	**kf42**	**FW**	KEY_F(42), 0462, Sent by function key f42
key_f43	**kf43**	**FX**	KEY_F(43), 0463, Sent by function key f43
key_f44	**kf44**	**FY**	KEY_F(44), 0464, Sent by function key f44
key_f45	**kf45**	**FZ**	KEY_F(45), 0465, Sent by function key f45
key_f46	**kf46**	**Fa**	KEY_F(46), 0466, Sent by function key f46
key_f47	**kf47**	**Fb**	KEY_F(47), 0467, Sent by function key f47
key_f48	**kf48**	**Fc**	KEY_F(48), 0470, Sent by function key f48
key_f49	**kf49**	**Fd**	KEY_F(49), 0471, Sent by function key f49
key_f50	**kf50**	**Fe**	KEY_F(50), 0472, Sent by function key f50
key_f51	**kf51**	**Ff**	KEY_F(51), 0473, Sent by function key f51
key_f52	**kf52**	**Fg**	KEY_F(52), 0474, Sent by function key f52
key_f53	**kf53**	**Fh**	KEY_F(53), 0475, Sent by function key f53
key_f54	**kf54**	**Fi**	KEY_F(54), 0476, Sent by function key f54
key_f55	**kf55**	**Fj**	KEY_F(55), 0477, Sent by function key f55
key_f56	**kf56**	**Fk**	KEY_F(56), 0500, Sent by function key f56
key_f57	**kf57**	**Fl**	KEY_F(57), 0501, Sent by function key f57
key_f58	**kf58**	**Fm**	KEY_F(58), 0502, Sent by function key f58
key_f59	**kf59**	**Fn**	KEY_F(59), 0503, Sent by function key f59
key_f60	**kf60**	**Fo**	KEY_F(60), 0504, Sent by function key f60
key_f61	**kf61**	**Fp**	KEY_F(61), 0505, Sent by function key f61
key_f62	**kf62**	**Fq**	KEY_F(62), 0506, Sent by function key f62
key_f63	**kf63**	**Fr**	KEY_F(63), 0507, Sent by function key f63
key_find	**kfnd**	**@0**	KEY_FIND, 0552, Sent by find key
key_help	**khlp**	**%1**	KEY_HELP, 0553, Sent by help key
key_home	**khome**	**kh**	KEY_HOME, 0406, Sent by home key
key_ic	**kich1**	**kI**	KEY_IC, 0513, Sent by ins-char/enter ins-mode key
key_il	**kil1**	**kA**	KEY_IL, 0511, Sent by insert-line key
key_left	**kcub1**	**kl**	KEY_LEFT, 0404, Sent by terminal left-arrow key
key_ll	**kll**	**kH**	KEY_LL, 0533, Sent by home-down key
key_mark	**kmrk**	**%2**	KEY_MARK, 0554, Sent by mark key
key_message	**kmsg**	**%3**	KEY_MESSAGE, 0555, Sent by message key
key_move	**kmov**	**%4**	KEY_MOVE, 0556, Sent by move key
key_next	**knxt**	**%5**	KEY_NEXT, 0557, Sent by next-object key
key_npage	**knp**	**kN**	KEY_NPAGE, 0522, Sent by next-page key
key_open	**kopn**	**%6**	KEY_OPEN, 0560, Sent by open key
key_options	**kopt**	**%7**	KEY_OPTIONS, 0561, Sent by options key
key_ppage	**kpp**	**kP**	KEY_PPAGE, 0523, Sent by previous-page key
key_previous	**kprv**	**%8**	KEY_PREVIOUS, 0562, Sent by previous-object key
key_print	**kprt**	**%9**	KEY_PRINT, 0532, Sent by print or copy key
key_redo	**krdo**	**%0**	KEY_REDO, 0563, Sent by redo key
key_reference	**kref**	**&1**	KEY_REFERENCE, 0564, Sent by ref(erence) key
key_refresh	**krfr**	**&2**	KEY_REFRESH, 0565, Sent by refresh key
key_replace	**krpl**	**&3**	KEY_REPLACE, 0566, Sent by replace key
key_restart	**krst**	**&4**	KEY_RESTART, 0567, Sent by restart key
key_resume	**kres**	**&5**	KEY_RESUME, 0570, Sent by resume key
key_right	**kcuf1**	**kr**	KEY_RIGHT, 0405, Sent by terminal right-arrow key
key_save	**ksav**	**&6**	KEY_SAVE, 0571, Sent by save key

key_sbeg	**kBEG**	**&9**	KEY_SBEG, 0572, Sent by shifted beginning key
key_scancel	**kCAN**	**&0**	KEY_SCANCEL, 0573, Sent by shifted cancel key
key_scommand	**kCMD**	***1**	KEY_SCOMMAND, 0574, Sent by shifted command key
key_scopy	**kCPY**	***2**	KEY_SCOPY, 0575, Sent by shifted copy key
key_screate	**kCRT**	***3**	KEY_SCREATE, 0576, Sent by shifted create key
key_sdc	**kDC**	***4**	KEY_SDC, 0577, Sent by shifted delete-char key
key_sdl	**kDL**	***5**	KEY_SDL, 0600, Sent by shifted delete-line key
key_select	**kslt**	***6**	KEY_SELECT, 0601, Sent by select key
key_send	**kEND**	***7**	KEY_SEND, 0602, Sent by shifted end key
key_seol	**kEOL**	***8**	KEY_SEOL, 0603, Sent by shifted clear-line key
key_sexit	**kEXT**	***9**	KEY_SEXIT, 0604, Sent by shifted exit key
key_sf	**kind**	**kF**	KEY_SF, 0520, Sent by scroll-forward/down key
key_sfind	**kFND**	***0**	KEY_SFIND, 0605, Sent by shifted find key
key_shelp	**kHLP**	**#1**	KEY_SHELP, 0606, Sent by shifted help key
key_shome	**kHOM**	**#2**	KEY_SHOME, 0607, Sent by shifted home key
key_sic	**kIC**	**#3**	KEY_SIC, 0610, Sent by shifted input key
key_sleft	**kLFT**	**#4**	KEY_SLEFT, 0611, Sent by shifted left-arrow key
key_smessage	**kMSG**	**%a**	KEY_SMESSAGE, 0612, Sent by shifted message key
key_smove	**kMOV**	**%b**	KEY_SMOVE, 0613, Sent by shifted move key
key_snext	**kNXT**	**%c**	KEY_SNEXT, 0614, Sent by shifted next key
key_soptions	**kOPT**	**%d**	KEY_SOPTIONS, 0615, Sent by shifted options key
key_sprevious	**kPRV**	**%e**	KEY_SPREVIOUS, 0616, Sent by shifted prev key
key_sprint	**kPRT**	**%f**	KEY_SPRINT, 0617, Sent by shifted print key
key_sr	**kri**	**kR**	KEY_SR, 0521, Sent by scroll-backward/up key
key_sredo	**kRDO**	**%g**	KEY_SREDO, 0620, Sent by shifted redo key
key_sreplace	**kRPL**	**%h**	KEY_SREPLACE, 0621, Sent by shifted replace key
key_sright	**kRIT**	**%i**	KEY_SRIGHT, 0622, Sent by shifted right-arrow key
key_srsume	**kRES**	**%j**	KEY_SRSUME, 0623, Sent by shifted resume key
key_ssave	**kSAV**	**!1**	KEY_SSAVE, 0624, Sent by shifted save key
key_ssuspend	**kSPD**	**!2**	KEY_SSUSPEND, 0625, Sent by shifted suspend key
key_stab	**khts**	**kT**	KEY_STAB, 0524, Sent by set-tab key
key_sundo	**kUND**	**!3**	KEY_SUNDO, 0626, Sent by shifted undo key
key_suspend	**kspd**	**&7**	KEY_SUSPEND, 0627, Sent by suspend key
key_undo	**kund**	**&8**	KEY_UNDO, 0630, Sent by undo key
key_up	**kcuu1**	**ku**	KEY_UP, 0403, Sent by terminal up-arrow key
keypad_local	**rmkx**	**ke**	Out of "keypad-transmit" mode
keypad_xmit	**smkx**	**ks**	Put terminal in "keypad-transmit" mode
lab_f0	**lf0**	**l0**	Labels on function key f0 if not f0
lab_f1	**lf1**	**l1**	Labels on function key f1 if not f1
lab_f2	**lf2**	**l2**	Labels on function key f2 if not f2
lab_f3	**lf3**	**l3**	Labels on function key f3 if not f3
lab_f4	**lf4**	**l4**	Labels on function key f4 if not f4
lab_f5	**lf5**	**l5**	Labels on function key f5 if not f5
lab_f6	**lf6**	**l6**	Labels on function key f6 if not f6
lab_f7	**lf7**	**l7**	Labels on function key f7 if not f7
lab_f8	**lf8**	**l8**	Labels on function key f8 if not f8
lab_f9	**lf9**	**l9**	Labels on function key f9 if not f9
lab_f10	**lf10**	**la**	Labels on function key f10 if not f10
label_off	**rmln**	**LF**	Turn off soft labels
label_on	**smln**	**LO**	Turn on soft labels
meta_off	**rmm**	**mo**	Turn off "meta mode"
meta_on	**smm**	**mm**	Turn on "meta mode" (8th bit)
newline	**nel**	**nw**	NEWLINE (behaves like **cr** followed by **lf**)
pad_char	**pad**	**pc**	Pad character (rather than null)
parm_dch	**dch**	**DC**	Delete #1 chars (G*)
parm_delete_line	**dl**	**DL**	Delete #1 lines (G*)
parm_down_cursor	**cud**	**DO**	Move cursor down #1 lines. (G*)
parm_ich	**ich**	**IC**	Insert #1 blank chars (G*)

parm_index	**indn**	**SF**	Scroll forward #1 lines. (G)
parm_insert_line	**il**	**AL**	Add #1 new blank lines (G*)
parm_left_cursor	**cub**	**LE**	Move cursor left #1 spaces (G)
parm_right_cursor	**cuf**	**RI**	Move cursor right #1 spaces. (G*)
parm_rindex	**rin**	**SR**	Scroll backward #1 lines. (G)
parm_up_cursor	**cuu**	**UP**	Move cursor up #1 lines. (G*)
pkey_key	**pfkey**	**pk**	Prog funct key #1 to type string #2
pkey_local	**pfloc**	**pl**	Prog funct key #1 to execute string #2
pkey_xmit	**pfx**	**px**	Prog funct key #1 to xmit string #2
plab_norm	**pln**	**pn**	Prog label #1 to show string #2
print_screen	**mc0**	**ps**	Print contents of the screen
prtr_non	**mc5p**	**pO**	Turn on the printer for #1 bytes
prtr_off	**mc4**	**pf**	Turn off the printer
prtr_on	**mc5**	**po**	Turn on the printer
repeat_char	**rep**	**rp**	Repeat char #1 #2 times (G*)
req_for_input	**rfi**	**RF**	Send next input char (for ptys)
reset_1string	**rs1**	**r1**	Reset terminal completely to sane modes
reset_2string	**rs2**	**r2**	Reset terminal completely to sane modes
reset_3string	**rs3**	**r3**	Reset terminal completely to sane modes
reset_file	**rf**	**rf**	Name of file containing reset string
restore_cursor	**rc**	**rc**	Restore cursor to position of last **sc**
row_address	**vpa**	**cv**	Vertical position absolute (G)
save_cursor	**sc**	**sc**	Save cursor position
scroll_forward	**ind**	**sf**	Scroll text up
scroll_reverse	**ri**	**sr**	Scroll text down
set_attributes	**sgr**	**sa**	Define the video attributes #1-#9 (G)
set_left_margin	**smgl**	**ML**	Set soft left margin
set_right_margin	**smgr**	**MR**	Set soft right margin
set_tab	**hts**	**st**	Set a tab stop in all rows, current column
set_window	**wind**	**wi**	Current window is lines #1-#2 cols #3-#4 (G)
tab	**ht**	**ta**	Move the cursor to the next 8 space hardware tab stop
to_status_line	**tsl**	**ts**	Go to status line, col #1 (G)
underline_char	**uc**	**uc**	Underscore one char and move past it
up_half_line	**hu**	**hu**	Half-line up (reverse 1/2 line-feed)
xoff_character	**xoffc**	**XF**	X-off character
xon_character	**xonc**	**XN**	X-on character

SAMPLE ENTRY

The following entry, which describes the Concept 100 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
concept100 | c100| concept | c104 | c100-4p | concept 100,
        am, db, eo, in, mir, ul, xenl, cols#80, lines#24, pb#9600, vt#8,
        bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew, cr=^M$<9>,
        cub1=^H, cud1=^J, cuf1=\E=, cup=\Ea%p1%' '%+%c%p2%' '%+%c, cuu1=\E;,
        cvvis=\EW, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
        ed=\E^C$<16*>, el=\E^U$<16>, flash=\Ek$<20>\EK, ht=\t$<8>,
        il1=\E^R$<3*>, ind=^J, .ind=^J$<9>, ip=$<16*>,
        is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
        kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;, kf1=\E5,
        kf2=\E6, kf3=\E7, khome=\E?, prot=\EI,
        rep=\Er%p1%c%p2%' '%+%c$<.2*>, rev=\ED,
        rmcup=\Ev\s\s\s\s$<6>\Ep\r\n, rmir=\E\0, rmkx=\Ex,
        rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\0,
        smcup=\EU\Ev\s\s8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
        smul=\EG,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with # are taken as comment lines. Capabilities in **terminfo** are of three types: boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or particular features, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (that is, an automatic RETURN and LINEFEED when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character # and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value **80** for the Concept. The value may be specified in decimal, octal or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an '=', and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in '**el=\EK$<3>**', and padding characters are supplied by **tputs**( ) (see **curses**(3V)) to provide this delay. The delay can be either a number, for example, **20**, or a number followed by an * (for example, **3***), a / (for example, **5/**), or both (for example, **10*/**). A * indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has **in** and the software uses it.) When a * is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.) A / indicates that the padding is mandatory. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

\E, \e	map to ESC
^X	maps to CTRL-*X* for any appropriate character *X*
\n	maps to NEWLINE
\l	maps to LINEFEED
\r	maps to RETURN
\t	maps to TAB
\b	maps to BACKSPACE
\f	maps to FORMFEED
\s	maps to SPACE
\0	maps to NUL

(**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret), \ (backslash), : (colon), and , (comma) may be given as \^, \\, \:, and \, respectively.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above. Note: capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with some *curses*-based application to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or bugs in the application. To test a new terminal description, set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/share/lib/terminfo**. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to insert 16 lines into the middle of a full screen at 9600 baud. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as Tektronix 4010 series, as well as hard-copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be RETURN, CTRL-M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**. If the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over; for example, you would not normally use **cuf1**=\s because the SPACE would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the **terminfo** file usually assumes that this is on; that is, **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (NEWLINE). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the model 33 teletype is described as

```
33 | tty33 | tty | model 33 teletype,
        bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM–3 is described as

> **adm3 | lsi adm3,**
> **am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,**
> **cud1=^J, ind=^J, lines#24,**

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf**(3V)-like escapes (**%x**) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it in the manner of a Reverse Polish Notation (postfix) calculator. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Binary operations are in postfix form with the operands in the usual order. That is, to get x–5 one would use '**%gx%{5}%–**'.

The % encodings have the following meanings:

%%	outputs %
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	as in **printf**(3V), flags are [–+#] and SPACE
%c	print **pop**( ) gives **%c**
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to **pop**( )
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push **strlen(pop( ))**
%+ %– %* %/ **%m**	arithmetic (**%m** is mod): **push(pop() op pop())**
%& %| %^	bit operations: **push(pop() op pop())**
%= %> %<	logical operations: **push(pop() op pop())**
%A %O	logical operations: and, or
%! %~	unary operations: **push(op pop())**
%i	(for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present
%?*expr*%**t***thenpart*%**e***elsepart*%;	if-then-else, '%**e***elsepart*' is optional; else-if's are possible in Algol 68: **%?** c_1 **%t** b_1 **%e** c_2 **%t** b_2 **%e** c_3 **%t** b_3 **%e** c_4 **%t** b_4 **%e** b_5**%;** c_i are conditions, b_i are bodies.

If the '–' flag is used with '**%[doxXs]**', then a colon (:) must be placed between the '%' and the '–' to differentiate the flag from the binary '%–' operator, for example, '**%:–16.16s**'.

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note: the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is:

```
cup=\E&a%p2%2.2dc%p1%2.2dY$<6>
```

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, '**cup=**^T%p1%c%p2%c'. Terminals which use **%c** need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set tty modes so that TAB characters are never expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus '**cup=**\E=%p1%'\s'%+%c%p2%'\s'%+%c'. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note: the home position is the same as addressing to (**0,0**): to the top left corner of the screen, not of memory. (Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the terminal has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the Tektronix 4025.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as '**il1**'; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as '**dl1**'; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using **terminfo**. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type '**abc def**' using local cursor motions (not SPACE characters) between the **abc** and the **def**. Then position the cursor before the **abc** and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped blanks) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals which have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an "insert mode" and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a TAB character after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode* (see **curses**(3V)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blanks on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many blanks are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Micro-Term MIME, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra-bright), **dim** (dim or half-bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse-video), **sgr0** (turn off all attribute modes), **smacs** (enter alternate-character-set mode), and **rmacs** (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes. If a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m

p3	reverse	\E[0;4m
p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Note: each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, since this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be '**\E[0;3;5m**'. The terminal does not have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N** depending on whether it is off or on. If all modes were to be turned on, the sequence would be '**\E[0;3;4;5;7;8m^N**'.

Now look at when different sequences are output. For example, '**;3**' is output when either '**p2**' or '**p6**' is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	**when to output**	**terminfo translation**
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;5%;%?%p1%p5%
|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note: it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, . . ., f63, the codes they send can be given as **kf0**, **kf1**, . . . , **kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0**, **lf1**, . . . , **lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (BACKSPACE), **ktbc** (clear all tab stops), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. A string to program their soft-screen labels can be given as **pln**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** executes the string by the terminal in local mode; and **pfx** transmits the string to the computer. The capabilities **nlab**, **lw** and **lh** define how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Tabs and Initialization

If the terminal has hardware tab stops, the command to advance to the next tab stop can be given as **ht** (usually CTRL-I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that TAB characters are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tab stops which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tab stops are set to. This is normally used by '**tput init**' (see **tput**(1V)) to determine whether to set the mode for hardware TAB expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the terminal; **iprog**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tab stops using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**(1V).

Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from **/usr/share/lib/tabset/***; however, the recommended method is to use the initialization and reset strings.) These strings are output by '**tput reset**', which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tab stops than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hard-copy terminals, and are used by '**tput init**' to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit H19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as TAB, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	VT100+ character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	–
diamond	‘
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

glyph name	VT100+ char	new tty char
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in '**acsc=lRmFkTjGq\,x.**'.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually CTRL-L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, '**tparm(repeat_char, 'x', 10)**' is the same as '**xxxxxxxxxx**'.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. On some UNIX systems, when the environment variable CC is set to a single-character value, all occurrences of the prototype character are replaced with that character.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch**, **dialup**, **patch**, and **network**, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to **virtual** terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not **^S** and **^Q** (CTRL-S and CTRL-Q, respectively), they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Special Cases

The working model used by **terminfo** fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by **terminfo**. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the **terminfo** model implemented.

Terminals which can not display tilde (˜) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a LINEFEED immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive TAB characters). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie" therefore, to erase standout mode, it is instead necessary to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or CTRL-C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for CTRL-C.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx***@** to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
        rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

FILES

/usr/share/lib/terminfo/?/*
compiled terminal description database

/usr/share/lib/tabset/* tab stop settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tab stops)

SEE ALSO

tput(1V), **curses**(3V), **printf**(3V), **term**(5V), **captoinfo**(8V), **infocmp**(8V), **tic**(8V)

WARNING

As described in the **Tabs and Initialization** section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a **curses**(3V) program is run. An available mechanism for outputting such strings is **tput init** (see **tput**(1V)).

Tampering with entries in **/usr/share/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

NAME

toc – table of contents of optional clusters in Application SunOS and Developer's Toolkit

SYNOPSIS

/usr/lib/load/toc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **toc** file contains information specifying the organization of the optional clusters in Application SunOS and Developer's Toolkit on the Sun386i distribution media. For each cluster, a single line should be present with the following information:

cluster name
set containing the cluster (Application SunOS or Developer's Toolkit)
size of the cluster (in kilobytes)
diskette volume of the cluster in the set (for loading from 3.5" diskette)
tape and file number of the cluster (for loading from 1/4" tape)

Items are separated by a ':'.

Cluster names can contain any printable character other than a ':', space, tab, or newline character. The set containing the cluster is specified by an 'A' for Application SunOS or 'D' for Developer's Toolkit. The diskette volume is the number of the diskette within the diskette set on which the cluster begins. The tape and file number specifies the tape and file position of the cluster on the tape.

EXAMPLE

The following is an example to the **toc** file.

```
accounting:A:55:14:1@12
advanced_admin:A:628:14:1@4
audit:A:144:14:1@8
comm:A:312:13:1@9
disk_quotas:A:56:14:1@11
doc_prep:A:790:13:1@10
extended_commands:A:276:13:1@5
games:A:2351:19:1@17
mail_plus:A:135:14:1@7
man_pages:A:5586:16:1@14
name_server:A:339:14:1@13
networking_plus:A:610:13:1@6
old:A:131:14:1@16
plot:A:227:14:1@14
spellcheck:A:455:13:1@2
sysV_commands:A:2505:14:1@3
base_devel:D:5389:1:2@2
plot_devel:D:247:5:2@3
sccs:D:328:5:2@4
sunview_devel:D:1768:5:2@5
sysV_devel:D:4287:3:2@6
proflibs:D:4755:4:2@7
config:D:3065:6:2@8
```

The first line specifies that the **accounting** cluster is part of Application SunOS and requires 55 kilobytes of disk storage. In the diskette distribution, it begins on diskette 14 of Application SunOS optional clusters. In the tape distribution, it can be found on file 12 of tape 1. The last line specifies that the *config* cluster is part of Developer's Toolkit and requires 3065 kilobytes of disk storage. In the diskette distribution, it begin on diskette 6 of Developer's Toolkit. In the tape distribution, it can be found on file 8 of tape 2.

FILES

/usr/lib/load/toc

SEE ALSO

cluster(1) **load**(1) **unload**(1)

NAME

translate – input and output files for system message translation

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

These files are used by **syslogd**(8) to translate systems messages. The input file is used to map system messages (in **printf**(3V) format strings) to numbers. This number is then used to locate a new string in the output file.

An initial part of each line in the input file may specify that the message should be suppressed. Recognized suppression specifications are:

(NONE) **Suppress the message always.**
(n) **Allow only one message every n seconds. ((10) for example).**
() **Do not suppress the message. This can be used in a message that begins with a '('.**

Note that the message suppression specification is optional. If not present, the message is not suppressed.

Each line in the output file translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message. The order of parameters passed from the input message can be changed, by replacing the % of a format phrase with a %*num*$ where *num* is a digit string. For example, if *num* is 2, the second parameter on the input file line will be used. The value of *num* can be from 1 to the number of parameters in the input message.

If a string is translated to a number that is not found in the output file, the message is suppressed.

EXAMPLES

An example input file:

```
$quote "
1       "(NONE)(1) logopen test code: %s\n"
2       "(10)(2) logopen test code: %s\n"
3       "()(3) logopen test code: %s\n"
4       "()(4) logopen test code: %s\n"
5       "(10)(5) logopen testcode: %s * 100\n"
6       "(10)(6) logopen testcode: %s * 100\n"
7       "(10)(7) logopen testcode: %s * 100\n"
8       "(10)%s: %s\n"
9       "(10)\n%s: write failed, file system is full\n"
10      "(10)NFS server %s not responding still trying\n"
11      "(10)NFS %s failed for server %s: %s\n"
12      "(10)NFS server %s ok\n"
13      "(NONE)\n%s: write failed, file system is full\n"
14      "(10)NFS server %s not responding still trying\n"
15      "(100)NFS %s failed for server %s: %s\n"
```

An example output file:

```
$quote "
1    "TRANSLATION:(1) logopen test code: %s\n"
2    "TRANSLATION: (2) logopen test code: %s IS REALLY\n"
3    "TRANSLATION: (3) logopen test code: %s\n"
4    "TRANSLATION: (4) logopen test code: %s\n"
5    "TRANSLATION: (5) logopen testcode: %s * 100\n"
6    "TRANSLATION: (6) logopen testcode: %s * 100\n"
7    "TRANSLATION: (7) logopen testcode: %s * 100\n"
8    "TRANSLATION: %s: %s\n"
9    "TRANSLATION: \n%s: write failed, file system is full\n"
10   "TRANSLATION: NFS server %s not responding still trying\n"
11   "TRANSLATION: NFS %s failed for server %s: %s\n"
12   "TRANSLATION: NFS server %s ok\n"
13   "Out of disk on file system %s\n"
14   "Network file server %s not ok. Check your cable\n"
15   "Network file server %2$s down (%1$s, %3$s)\n"
```

SEE ALSO

syslogd(8)

NAME

ttytab, ttys – terminal initialization data

DESCRIPTION

The **/etc/ttytab** file contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the **getttyent**(3) library routines. There is one line in **/etc/ttytab** file per special file.

The **/etc/ttys** file should not be edited; it is derived from **/etc/ttytab** by **init(8)** at boot time, and is only included for backward compatibility with programs that may still require it.

Fields are separated by TAB and/or SPACE characters. Some fields may contain more than one word and should be enclosed in double quotes. Blank lines and comments can appear anywhere in the file; comments are delimited by '#' and NEWLINE. Unspecified fields default to NULL. The first field is the terminal's entry in the device directory, **/dev**. The second field of the file is the command to execute for the line, typically **getty**(8), which performs such tasks as baud-rate recognition, reading the login name, and calling **login**(1). It can be, however, any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted. The third field is the type of terminal normally connected to that tty line, as found in the **termcap**(5) data base file. The remaining fields set flags in the **ty_status** entry (see **getttyent**(3)) or specify a window system process that **init**(8) will maintain for the terminal line.

As flag values, the strings **on** and **off** specify whether **init** should execute the command given in the second field, while **secure** in addition to **on** allows "root" to login on this line. If the console is not marked "secure," the system prompts for the root password before coming up in single-user mode. **local** in addition to **on** indicates that the line is a "local" line; the modem control signals for this line, such as Carrier Detect, will be ignored. These flag fields should not be quoted. The string **window=** is followed by a quoted command string which **init** will execute before starting **getty**.

The flag **local** applies to terminals, and enables the software carrier mode in the kernel; the kernel ignores the state of carrier detect when opening the serial port. Alternately, if this field is set to any value other than **local**, this flag disables the software carrier mode in the kernel, so the state of the carrier detect is not ignored. This usually applies to modems. See **termio**(4).

If the line ends in a comment, the comment is included in the **ty_comment** field of the **ttyent** structure.

After changing the **/etc/ttytab** file, you must notify **init**(8) before those changes will take effect. To do this, use:

kill –1 1

EXAMPLES

Below is a sample **/etc/ttytab** file:

```
console "/usr/etc/getty std.1200"  vt100      on secure
ttyd0   "/usr/etc/getty d1200"     dialup     on          # 555-1234
ttyh0   "/usr/etc/getty std.9600"  hp2621-nl  on          # 254MC
ttyh1   "/usr/etc/getty std.9600"  plugboard  on          # John's office
ttyp0   none                       network
ttyp1   none                       network    off
ttyv0   "/usr/new/xterm -L :0"     vs100      on window="/usr/new/Xvs100 0"
console "/usr/etc/getty -n -s  std.9600" sun    on   secure
console "/usr/etc/getty -n -s -l std.9600" sun  on   secure
```

The first line permits "root" login on the console at 1200 baud, and indicates that the console is physically secure for single-user operation. The second line allows dialup at 1200 baud without "root" login, and the third and fourth lines allow login at 9600 baud with terminal types of **hp2621-nl** and **plugboard**, respectively. The fifth and sixth lines are examples of network pseudo-ttys, **ttyp0** and **ttyp1** for which **getty** should not be enabled. The seventh line shows a terminal emulator and window-system startup entry. The last two lines instruct **getty,** using the **−n** argument, to run the **logintool**(8) graphic login interface, and the **−s** argument instructing **logintool** to start **screenblank**(1) with a plain black screen. The **−l** (lower case L) argument instructs **logintool** to start **lockscreen**(1). **lockscreen** starts after 30 minutes; there is no way to change this interval.

FILES

/dev
/etc/ttys
/etc/ttytab

SEE ALSO

login(1), **ioctl**(2), **getttyent**(3), **termio**(4), **gettytab**(5), **termcap**(5), **getty**(8), **init**(8), **logintool**(8), **ttysoftcar**(8)

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in the system code; some data of these types are accessible to user code:

```
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#ifndef  _TYPES_
#define  _TYPES_

/*
 * Basic system types.
 */

#include <sys/sysmacros.h>

typedef  unsigned char    u_char;
typedef  unsigned short   u_short;
typedef  unsigned int     u_int;
typedef  unsigned long    u_long;
typedef  unsigned short   ushort;/* System V compatibility */
typedef  unsigned int     uint;/* System V compatibility */

#ifdef vax
typedef  struct     _physadr { int r[1]; } *physadr;
typedef  struct     label_t{
         int        val[14];
} label_t;
#endif
#ifdef mc68000
typedef  struct     _physadr { short r[1]; } *physadr;
typedef  struct     label_t{
         int        val[13];
} label_t;
#endif
#ifdef sparc
typedef struct  _physadr { int r[1]; } *physadr;
typedef struct label_t {
         int    val[2];
} label_t;
#endif
#ifdef i386
typedef struct      _physadr { short r[1]; } *physadr;
typedef struct      label_t {
         int        val[8];
} label_t;
```

```
#endif
typedef struct    _quad { long val[2]; } quad;
typedef long      daddr_t;
typedef char *    caddr_t;
typedef u_long    ino_t;
typedef long      swblk_t;
typedef int       size_t;
typedef long      time_t;
typedef short     dev_t;
typedef long      off_t;
typedef u_short   uid_t;
typedef u_short   gid_t;
typedef long      key_t;

#define NBBY      8       /* number of bits in a byte */
/*
 * Select uses bit masks of file descriptors in longs.
 * These macros manipulate such bit fields (the filesystem macros use chars).
 * FD_SETSIZE may be defined by the user, but the default here
 * should be >= NOFILE (param.h).
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE    256
#endif

typedef long      fd_mask;
#define NFDBITS         (sizeof(fd_mask) * NBBY)/* bits per mask */
#ifndef howmany
#ifdef sun386
#define howmany(x, y)   ((((u_int)(x))+(((u_int)(y))-1))/((u_int)(y)))
#else
#define howmany(x, y)   (((x)+((y)-1))/(y))
#endif
#endif

typedef struct fd_set {
        fd_mask  fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

typedef char *    addr_t;

#define FD_SET(n, p)    ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p)    ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p)  ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)      bzero((char *)(p), sizeof(*(p)))

#ifdef sparc
/*
 * routines that call setjmp have strange control flow graphs,
 * since a call to a routine that calls resume/longjmp will eventually
 * return at the setjmp site, not the original call site. This
 * utterly wrecks control flow analysis.
 */
```

```
extern int setjmp();
#pragma unknown_control_flow(setjmp)
#endif sparc

#endif   _TYPES_
```

The form *daddr_t* is used for disk addresses, see **fs**(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

adb(1), **lseek**(2V), **time**(3V), **fs**(5)

NAME

tzfile – time zone information

SYNOPSIS

#include <tzfile.h>

DESCRIPTION

The time zone information files used by **tzset** (see **ctime**(3V)) begin with bytes reserved for future use, followed by three four-byte values of type **long**, written in a "standard" byte order (the high-order byte of the value is written first). These values are, in order:

tzh_timecnt	The number of "transition times" for which data is stored in the file.
tzh_typecnt	The number of "local time types" for which data is stored in the file (must not be zero).
tzh_charcnt	The number of characters of "time zone abbreviation strings" stored in the file.

The above header is followed by *tzh_timecnt* four-byte values of type **long**, sorted in ascending order. These values are written in "standard" byte order. Each is used as a transition time (as returned by **gettimeofday**(2)) at which the rules for computing local time change. Next come *tzh_timecnt* one-byte values of type **unsigned char**; each one tells which of the different types of "local time" types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of *ttinfo* structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
        long          tt_gmtoff;
        int           tt_isdst;
        unsigned int  tt_abbrind;
};
```

Each structure is written as a four-byte value for *tt_gmtoff* of type **long**, in a standard byte order, followed by a one-byte value for *tt_isdst* and a one-byte value for *tt_abbrind*. In each structure, *tt_gmtoff* gives the number of seconds to be added to GMT, *tt_isdst* tells whether *tm_isdst* should be set by **localtime** (see **ctime**(3V)) and *tt_abbrind* serves as an index into the array of time zone abbreviation characters that follow the *ttinfo* structure(s) in the file.

localtime uses the first standard-time *ttinfo* structure in the file (or simply the first *ttinfo* structure in the absence of a standard-time structure) if either *tzh_timecnt* is zero or the time argument is less than the first transition time recorded in the file.

SEE ALSO

gettimeofday(2), **ctime**(3V)

NAME

ugid_alloc.range – range of user IDs and group IDs to allocate

SYNOPSIS

/etc/ugid_alloc.range

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/ugid_alloc.range** file, if it exists on the Network Information Service (NIS) master of the **passwd.byuid** map (or the **group.bygid** map for group IDs), specifies the user IDs and group IDs that can be allocated for the local NIS domain by the **uid_allocd**(8C) daemons. If the file does not exist, user IDs or group IDs may be allocated beginning at 100 and ending at 60,000; no user IDs or group IDs are allocated out of that range in any case. If the local NIS domain is not listed in this file, no user IDs or group IDs will be allocated. Otherwise, this file specifies ranges of user IDs or group IDs that may be allocated. The different NIS domains on a network can use identical copies of this file.

If a network has multiple NIS domains, each one will typically use ranges for its user IDs and group IDs that do not overlap with the other NIS domains, guaranteeing that user IDs and group IDs are unique throughout the network. Without guarantees of user ID and group ID uniqueness, network tools and services which rely on that uniqueness for security or authentication will not work as intended. Such services include NFS, except for the "Secure NFS," which has other solutions for security and authentication. Note: the required uniqueness could be guaranteed by mechanisms other than automatic allocation within manually configured ranges. For example, some sites can use a function of their employee numbers during manual user ID allocation, and coordinate group ID assignment verbally.

This file can contain blank lines. Comments begin with a '#' character and extend to the end of the current line. The first token on the line is an NIS domain name. It is separated from the second token by white space (SPACE or TAB characters). The second token is either *user* or *group*, indicating that the line specifies user ID or group ID ranges, respectively. The third token is a comma-separated list of user or group ID ranges in that domain. These ranges take two forms: a single number specifies just that ID, and two numbers separated by a dash specify all IDs starting at the first number and ending with the second.

For example, the following file would direct that the manufacturing department at a particular company use user IDs from 700 to 999 or 1200 to 1499. Accounts created by tools in the NIS domain for manufacturing would use a user ID in those ranges, and those user accounts could safely be added to one of the other NIS domains if desired (by manually transferring NIS map data between the domains). Group IDs are allocated only within the administration domain.

```
# Three departments share our site's network, and each has its
# own Ethernet and master server connected with IP routers.
# This file sets the user ID ranges assigned to each department.
# Groups are defined by the administration group only.
YP.admin.company.com            user    500-699
YP.manufacturing.company.com    user    700-999
YP.engineering.company.com      user    100-499,1000-1199
YP.manufacturing.company.com    user    1200-1499
YP.admin.company.com            group   100-60000
```

SEE ALSO

passwd(5), **group**(5), **uid_allocd**(8C)

BUGS

There is a limit of forty ranges for each domain; more ranges are silently ignored.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME
updaters – configuration file for NIS updating

SYNOPSIS
/var/yp/updaters

DESCRIPTION
The file **/var/yp/updaters** is a makefile (see **make**(1)) which is used for updating the Network Information Service (NIS) databases. Databases can only be updated in a secure network, that is, one that has a **publickey**(5) database. Each entry in the file is a make target for a particular NIS database. For example, if there is an NIS database named **passwd.byname** that can be updated, there should be a **make** target named **passwd.byname** in the **updaters** file with the command to update the file.

The information necessary to make the update is passed to the update command through standard input. The information passed is described below (all items are followed by a NEWLINE, except for 4 and 6)

- Network name of client wishing to make the update (a string)
- Kind of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status **YPERR_ACCESS**. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid NIS error code described in **<rpcsvc/ypclnt.h>**.

FILES
/var/yp/updaters

SEE ALSO
make(1), **ypupdate**(3N), **publickey**(5), **ypupdated**(8C)

NOTES
The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

utmp, wtmp, lastlog – login records

SYNOPSIS

#include <utmp.h>
#include <lastlog.h>

DESCRIPTION

utmp file

The **utmp** file records information about who is currently using the system. The file is a sequence of **utmp** structure entries. That structure is defined in **<utmp.h>**, and contains the following members:

ut_line Character array containing the name of the terminal on which the user logged in.

ut_name Character array containing the name of the user who logged in.

ut_host Character array containing the name of the host from which the user remotely logged in, if they logged in from another host; otherwise, a null string.

ut_time **long** containing the time at which the user logged in, in seconds since 00:00 GMT, January 1, 1970.

Whenever a user logs in, **login**(1) fills in the entry in **/etc/utmp** for the terminal on which the user logged in. When they log out, **init**(8) clears that entry by setting **ut_name** and **ut_host** to null strings and **ut_time** to the time at which the user logged out.

Some window systems will make entries in **utmp** for terminal emulation windows running shells, so that library routines such as **getlogin** will work correctly in that window. These entries do not directly represent logged-in users; they are associated with a user who has already logged into the system on another terminal. These entries generally have a **ut_line** field that refers to a pseudo-terminal, and a **ut_host** field that is a null string. The macro **nonuser**, defined in **<utmp.h>**, takes a pointer to a **utmp** structure as an argument and, if the entry has a **ut_line** field that refers to a pseudo-terminal, and a **ut_host** field that is a null string, will return 1; otherwise, it will return 0. This can be used by programs that print information about logged-in users if they should not list entries made for logged-in users' additional windows.

wtmp file

The **wtmp** file records all logins and logouts. It also consists of a sequence of **utmp** entries.

Whenever a user logs in, **login** appends a record identical to the record it placed in **utmp** to the end of **/var/adm/wtmp.** Whenever a user logs out, **init** appends a record with **ut_line** equal to the terminal that the user was logged in on, **ut_name** and **ut_host** null, and **ut_time** equal to the time at which the user logged out.

When the system is shut down, **init** appends a record with a **ut_line** of ~, a **ut_name** of **shutdown**, a null **ut_host**, and a **ut_time** equal to the time at which the shutdown occurred. When the system is rebooted, **init** appends a record with a **ut_line** of ~, a **ut_name** of **reboot**, a null **ut_host**, and a **ut_time** equal to the time at which **init** wrote the record.

When the **date** command is used to change the system-maintained time, **date** appends a record with a **ut_line** of **|**, **ut_name** and **ut_host** null, and **ut_time** equal to the system time before the change, and then appends a record with a **ut_line** of **{**, **ut_name** and **ut_host** null, and **ut_time** equal to the system time after the change.

None of the programs that maintain **wtmp** create the file, so that if record-keeping is to be enabled, it must be created by hand as a zero-length file, and if it is removed, record-keeping is turned off. It is summarized by **ac**(8).

As **wtmp** is appended to whenever a user logs in or out, it should be truncated periodically so that it does not consume all the disk space on its file system.

lastlog file

The **lastlog** file records the most recent login-date for every user logged in. The file is a sequence of **lastlog** structure entries. That structure is defined in **<lastlog.h>**, and contains the following members:

ll_time **long** containing the time at which the user logged in, in seconds since 00:00 GMT, January 1, 1970.

ll_line Character array containing the name of the terminal on which the user logged in.

ll_host Character array containing the name of the host from which the user remotely logged in, if they logged in from another host; otherwise, a null string.

When reporting (and updating) the most recent login date, **login** performs an **lseek**(2V) to a byte-offset in **/var/adm/lastlog** corresponding to the userid. Because the count of userids may be high, whereas the number actual users may be small within a network environment, the bulk of this file may never be allocated by the file system even though an offset may appear to be quite large. Although **ls**(1V) may show it to be large, chances are that this file need not be truncated. **du**(1V) will report the correct (smaller) amount of space actually allocated to it.

SYSTEM V DESCRIPTION

For XPG2 conformance, the XPG2 private **utmp** structure is preserved for use by compliant applications that specifically use the **utmp structure.** The structure is defined in **/usr/xpg2include/utmp.h**. Note: this structure definition was removed in XPG3, and will be removed in a future SunOS release. Applications using the XPG2 **utmp** structure must do so on an application private basis.

FILES

/etc/utmp
/var/adm/wtmp
/var/adm/lastlog

SEE ALSO

login(1), **who**(1), **ac**(8), **init**(8)

NAME

uuencode – format of an encoded uuencode file

DESCRIPTION

Files output by **uuencode**(1C) consist of a header line, followed by a number of body lines, and a trailer line. **uudecode** (see **uuencode**(1C)) will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters '**begin** '. The word **begin** is followed by a mode (in octal), and a string which names the remote file. Spaces separate the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing NEWLINE). These consist of a character count, followed by encoded characters, followed by a NEWLINE. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a SPACE to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII SPACE.

The trailer line consists of **end** on a line by itself.

SEE ALSO

mail(1), **uucp**(1C), **uuencode**(1C), **uusend**(1C)

NAME

vfont – font formats

SYNOPSIS

#include <vfont.h>

DESCRIPTION

The fonts used by the window system and printer/plotters have the following format. Each font is in a file, which contains a header, an array of character description structures, and an array of bytes containing the bit maps for the characters. The header has the following format:

```
struct header {
        short           magic;          /* Magic number VFONT_MAGIC */
        unsigned short  size;           /* Total # bytes of bitmaps */
        short           maxx;           /* Maximum horizontal glyph size */
        short           maxy;           /* Maximum vertical glyph size */
        short           xtend;          /* (unused) */
};
#define VFONT_MAGIC                     0436
```

maxx and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. (A glyph is just a printed representation of a character, in a particular size and font.) The **size** is the total size of the bit maps for the characters in bytes. The *xtend* field is not currently used.

After the header is an array of **NUM_DISPATCH** structures, one for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
        unsigned short  addr;                   /* &(glyph) - &(start of bitmaps) */
        short           nbytes;                 /* # bytes of glyphs (0 if no glyph) */
        char            up, down, left, right;  /* Widths from baseline point */
        short           width;                  /* Logical width, used by troff */
};
#define NUM_DISPATCH                            256
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the bit maps to where the character's bit map begins. The *up* , *down*, *left*, and *right* fields are offsets from the base point of the glyph to the edges of the rectangle which the bit map represents. (The imaginary ''base point'' is a point which is vertically on the ''base line'' of the glyph (the bottom line of a glyph which does not have a descender) and horizontally near the left edge of the glyph; often 3 or so pixels past the left edge.) The bit map contains *up+down* rows of data for the character, each of which has *left+right* columns (bits). Each row is rounded up to a number of bytes. The *width* field represents the logical width of the glyph in bits, and shows the horizontal displacement to the base point of the next glyph.

FILES

/usr/lib/vfont/*
/usr/lib/fonts/fixedwidthfonts/*

SEE ALSO

troff(1), **vfontinfo**(1), **vswap**(1)

BUGS

A machine-independent font format should be defined. The **shorts** in the above structures contain different bit patterns depending whether the font file is for use on a VAX or a Sun. The **vswap** program must be used to convert one to the other.

NAME

vgrindefs – vgrind's language definition data base

SYNOPSIS

/usr/lib/vgrindefs

DESCRIPTION

vgrindefs contains all language definitions for **vgrind**(1). The data base is very similar to **termcap**(5). Capabilities in **vgrindefs** are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list. Entries may continue onto multiple lines by giving a \ as the last character of a line. Lines starting with # are comments.

Capabilities

The following table names and describes each capability.

Name	Type	Description
ab	str	Regular expression for the start of an alternate form comment
ae	str	Regular expression for the end of an alternate form comment
bb	str	Regular expression for the start of a block
be	str	Regular expression for the end of a lexical block
cb	str	Regular expression for the start of a comment
ce	str	Regular expression for the end of a comment
id	str	String giving characters other than letters and digits that may legally occur in identifiers (default '_')
kw	str	A list of keywords separated by spaces
lb	str	Regular expression for the start of a character constant
le	str	Regular expression for the end of a character constant
oc	bool	Present means upper and lower case are equivalent
pb	str	Regular expression for start of a procedure
pl	bool	Procedure definitions are constrained to the lexical level matched by the 'px' capability
px	str	A match for this regular expression indicates that procedure definitions may occur at the next lexical level. Useful for lisp-like languages in which procedure definitions occur as subexpressions of defuns.
sb	str	Regular expression for the start of a string
se	str	Regular expression for the end of a string
tc	str	Use the named entry as a continuation of this one
tl	bool	Present means procedures are only defined at the top lexical level

Regular Expressions

vgrindefs uses regular expressions similar to those of **ex**(1) and **lex**(1). The characters '^', '$', ':', and '\' are reserved characters and must be 'quoted' with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$ The end of a line

^ The beginning of a line

\d A delimiter (space, tab, newline, start of line)

\a Matches any string of symbols (like '.*' in lex)

\p Matches any identifier. In a procedure definition (the 'pb' capability) the string that matches this symbol is used as the procedure name.

() Grouping

| Alternation

? Last item is optional

\e Preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) that can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like '(tramp|steamer)flies?' would match 'tramp', 'steamer', 'trampflies', or 'steamerflies'. Contrary to some forms of regular expressions, **vgrindef** alternation binds very tightly. Grouping parentheses are likely to be necessary in expressions involving alternation.

Keyword List

The keyword list is just a list of keywords in the language separated by spaces. If the 'oc' boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

EXAMPLE

The following entry, which describes the C language, is typical of a language entry.

```
C|c|the C programming language:\
        :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do double else enum\
        extern float for fortran goto if int long register return short\
        sizeof static struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to **vgrind**(1) as 'c' or 'C'.

FILES

/usr/lib/vgrindefs file containing terminal descriptions

SEE ALSO

troff(1), **vgrind**(1)

NAME

ypaliases – NIS aliases for sendmail

SYNOPSIS

/etc/ypaliases

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) aliases map with this text file. The **/etc/ypaliases** file has the same format as the **/etc/aliases** file described in **aliases**(5).

The text file for the NIS aliases map is stored in the **/etc/aliases** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/aliases** file. The local file is accessed first by programs such as **sendmail**(8), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/aliases** file can specify resources that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/aliases** file to specify aliases that are to be known only to the local system. Sun386i systems allow the **/etc/aliases** file on the NIS master to be used locally, creating the NIS aliases map with the **/etc/ypaliases** text file.

FILES

/etc/aliases
/etc/ypaliases

SEE ALSO

uucp(1C), **dbm**(3X), **aliases**(5), **newaliases**(8), **sendmail**(8)

System and Network Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypfiles – NIS database and directory structure

DESCRIPTION

The Network Information Service (NIS) uses a distributed, replicated database of **dbm** files contained in the **/var/yp** directory hierarchy on each NIS server. A **dbm** database consists of two files, created by calls to the **ndbm**(3) library package. One has the filename extension **.pag** and the other has the filename extension **.dir**. For instance, the database named **hosts.byname**, is implemented by the pair of files **hosts.byname.pag** and **hosts.byname.dir.**

A **dbm** database served by the NIS service is called an NIS *map*. An NIS *domain* is a subdirectory of **/var/yp** containing a set of NIS maps. Any number of NIS domains can exist. Each may contain any number of maps.

No maps are required by the NIS lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which the NIS service serves — if the map exists in a given domain, and a client asks about it, the NIS service will serve it. For a map to be accessible consistently, it must exist on all NIS servers that serve the domain. To provide data consistency between the replicated maps, an entry to run **ypxfr** periodically should be made in the super-user's **crontab** file on each server. More information on this topic is in **ypxfr**(8).

The NIS maps should contain two distinguished key-value pairs. The first is the key **YP_LAST_MODIFIED**, having as a value a ten-character ASCII order number. The order number should be the system time in seconds when the map was built. The second key is **YP_MASTER_NAME**, with the name of the NIS master server as a value. **makedbm**(8) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the NIS service, but the **ypserv** process will not be able to return values for "Get order number" or "Get master name" requests. See **ypserv**(8). In addition, values of these two keys are used by **ypxfr** when it transfers a map from a master NIS server to a slave. If **ypxfr** cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

The NIS maps must be generated and modified only at the master server. They are copied to the slaves using **ypxfr**(8) to avoid potential byte-ordering problems among the NIS servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The NIS database can be initially set up for both masters and slaves by using **ypinit**(8).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the NIS map and is propagated from the master to the slaves by running **/var/yp/Makefile.** All Sun-supplied maps have entries in **/var/yp/Makefile**; if you add an NIS map, edit this file to support the new map. The makefile uses **makedbm**(8) to generate the NIS map on the master, and **yppush**(8) to propagate the changed map to the slaves. **yppush** is a client of the map **ypservers**, which lists all the NIS servers. For more information on this topic, see **yppush**(8).

FILES

/var/yp
/var/yp/Makefile

SEE ALSO

dbm(3X), **makedbm**(8), **rpcinfo**(8C), **ypinit**(8), **ypmake**(8), **yppoll**(8), **yppush**(8), **ypserv**(8), **ypxfr**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypgroup – NIS group file

SYNOPSIS

/etc/ypgroup

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) group map with this text file. This file has the same format as the **/etc/group** file described in **group**(5).

The text file for the NIS group map is stored in the **/etc/group** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/group** file. The local file is accessed first by programs such as **groups**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed. The local **/etc/group** file can specify groups that are not available on a network-wide basis.

This implies that the NIS master cannot use the local **/etc/group** file to specify groups that are to be known only to the local system. Sun386i systems allow the **/etc/group** file on the NIS master to be used locally, creating the NIS group map from the **/etc/ypgroup** text file.

FILES

/etc/group
/etc/ypgroup

SEE ALSO

passwd(1), **su**(1V), **getgroups**(2V), **crypt**(3), **initgroups**(3), **group**(5), **group.adjunct**(5), **passwd**(5), **grpck**(8V)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yppasswd – NIS password file

SYNOPSIS

/etc/yppasswd

DESCRIPTION

Create the Network Information Service (NIS) password map with this text file. The format for **/etc/yppasswd** is the same as for the **/etc/passwd** file described in **passwd**(5).

The text file for the NIS password map is stored in the **/etc/passwd** file on the NIS master of an NIS domain. Other systems in a domain can also have a local **/etc/passwd** file. The local file is accessed first by programs such as **passwd**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/passwd** file can specify users that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/passwd** file to specify users that are to be known only to the local system. Sun386i systems allow the **/etc/passwd** file on the NIS master to be used locally, creating the NIS password map from the **/etc/yppasswd** text file.

FILES

/etc/passwd
/etc/yppasswd

SEE ALSO

login(1), **mail**(1), **passwd**(1), **crypt**(3), **getpwent**(3V), **group**(5), **passwd**(5), **passwd.adjunct**(5), **adduser**(8), **sendmail**(8), **vipw**(8)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypprintcap – NIS printer capability database

SYNOPSIS

/etc/ypprintcap

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) printcap map with this text file to centralize and simplify printer administration. The **/etc/ypprintcap** file has the same format as the **/etc/printcap** file described in **printcap**(5).

The text file for the NIS printcap map is stored in the **/etc/printcap** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/printcap** file. The local file is accessed first by programs such as **lpr**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/printcap** file can specify printers that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/printcap** file to specify printers that are to be known only to the local system. Sun386i systems allow the **/etc/printcap** file on the NIS master to be used locally, using the **/etc/ypprintcap** file to create the NIS printcap map.

FILES

/etc/printcap
/etc/ypprintcap

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **snap**(1), **stty**(1V), **plot**(3X), **ttcompat**(4M), **printcap**(5), **termcap**(5), **lpc**(8), **lpd**(8), **pac**(8)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME
intro – introduction to games and demos

DESCRIPTION
This section describes available games and demos.

LIST OF GAMES AND DEMOS

Name	Appears on Page	Description
adventure	**adventure**(6)	an exploration game
arithmetic	**arithmetic**(6)	provide drill in number facts
backgammon	**backgammon**(6)	the game of backgammon
banner	**banner**(6)	print large banner on printer
battlestar	**battlestar**(6)	a tropical adventure game
bcd	**bcd**(6)	convert to antique media
bdemos	**bdemos**(6)	demonstrate Sun Monochrome Bitmap Display
bdraw	**draw**(6)	interactive graphics drawing
bj	**bj**(6)	the game of black jack
boggle	**boggle**(6)	play the game of boggle
boggletool	**boggletool**(6)	play a game of boggle
bouncedemo	**graphics_demos**(6)	graphics demonstration programs
brotcube	**brotcube**(6)	rotate a simple cube
bsuncube	**bsuncube**(6)	view 3-D Sun logo
buttontest	**buttontest**(6)	demonstration and testing program for SunButtons
canfield	**canfield**(6)	Canfield solitaire card game
canfieldtool	**canfield**(6)	Canfield solitaire card game
canvas_demo	**sunview_demos**(6)	Window-System demonstration programs
cdplayer	**cdplayer**(6)	CD-ROM audio demo program
cdraw	**draw**(6)	interactive graphics drawing
cfscores	**canfield**(6)	Canfield solitaire card game
chess	**chess**(6)	the game of chess
chesstool	**chesstool**(6)	window-based front-end to chess program
ching	**ching**(6)	the book of changes and other cookies
colordemos	**colordemos**(6)	demonstrate Sun Color Graphics Display
craps	**craps**(6)	the game of craps
cribbage	**cribbage**(6)	the card game cribbage
cursor_demo	**sunview_demos**(6)	Window-System demonstration programs
dialtest	**dialtest**(6)	demonstration and testing program for SunDials
draw	**draw**(6)	interactive graphics drawing
factor	**factor**(6)	factor a number, generate large primes
fish	**fish**(6)	play "Go Fish"
flight	**gp_demos**(6)	demonstration programs for the Graphics Processor
fortune	**fortune**(6)	print a random, hopefully interesting, adage
framedemo	**graphics_demos**(6)	graphics demonstration programs
gaintool	**gaintool**(6)	audio control panel
gammontool	**gammontool**(6)	play a game of backgammon
gp_demos	**gp_demos**(6)	demonstration programs for the Graphics Processor
graphics_demos	**graphics_demos**(6)	graphics demonstration programs
hack	**hack**(6)	replacement for rogue
hangman	**hangman**(6)	computer version of the game hangman
hunt	**hunt**(6)	a multiplayer multiterminal game
jumpdemo	**graphics_demos**(6)	graphics demonstration programs
life	**life**(6)	John Conway's game of life
mille	**mille**(6)	play Mille Bornes
monop	**monop**(6)	Monopoly game

moo	**moo**(6)	guessing game
number	**number**(6)	convert Arabic numerals to English
play	**play**(6)	play audio files
ppt	**bcd**(6)	convert to antique media
primes	**factor**(6)	factor a number, generate large primes
primes	**primes**(6)	print all primes larger than some given number
quiz	**quiz**(6)	test your knowledge
rain	**rain**(6)	animated raindrops display
random	**random**(6)	select lines randomly from a file
raw2audio	**raw2audio**(6)	convert raw audio data to audio file format
record	**record**(6)	record an audio file
robots	**robots**(6)	fight off villainous robots
rotcvph	**rotcvph**(6)	rotate convex polyhedron
rotobj	**gp_demos**(6)	demonstration programs for the Graphics Processor
snake	**snake**(6)	display chase game
snscore	**snake**(6)	display chase game
soundtool	**soundtool**(6)	audio play/record tool
spheresdemo	**graphics_demos**(6)	graphics demonstration programs
suncoredemos	**suncoredemos**(6)	demonstrate SunCore Graphics Package
sunview_demos	**sunview_demos**(6)	Window-System demonstration programs
trek	**trek**(6)	trekkie game
vwcvph	**vwcvph**(6)	view convex polyhedron
worm	**worm**(6)	play the growing worm game
worms	**worms**(6)	animate worms on a display terminal
wump	**wump**(6)	the game of hunt the wumpus

NAME

adventure – an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type **quit**; to save a game for later resumption, type **suspend**.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+–x/] [*range*]

DESCRIPTION

arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (such as CTRL–C).

The first optional argument determines the kind of problem to be generated; '+', '–', '**x**', '/' respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +–.

range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game of backgammon

SYNOPSIS

backgammon [–] [**n r w b pr pw pb t***term* **s***filename*]

DESCRIPTION

backgammon lets you play backgammon against the computer or against a 'friend'. All commands only are one letter, so you don't need to type a carriage return, except at the end of a move. **backgammon** is mostly self documenting, so that a q **?** (question mark) will usually get some help. If you answer **y** when **backgammon** asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instruction on how to use **backgammon,** and a tutorial consisting of a practice game against the computer. A description of how to use **backgammon** can be obtained by answering **y** when it asks if you want instructions. The possible arguments for **backgammon** (most are unnecessary but some are very convenient) consist of:

n	don't ask for rules or instructions
r	player is red (implies n)
w	player is white (implies n)
b	two players, red and white (implies n)
pr	print the board before red's turn
pw	print the board before white's turn
pb	print the board before both player's turn
t*term*	terminal is type *term*, uses */etc/termcap*, otherwise uses the TERM environment variable.
s*filename*	recover previously saved game from *filename*. This can also be done by executing the saved file, that is, typing its name in as a command.

Arguments may be optionally preceded by a – sign. Several arguments may be concatenated together, but not after **s** or **t** arguments, since they can be followed by an arbitrary string. Any unrecognized arguments are ignored. An argument of a lone – gets a description of possible arguments.

If **term** has capabilities for direct cursor movement. **backgammon** 'fixes' the board after each move, so the board does not need to be reprinted, unless the screen suffers some horrendous malady. Also, any 'p' option will be ignored.

QUICK REFERENCE

When **backgammon** prompts by typing only your color, type a space or carriage return to roll, or

d	to double
p	to print the board
q	to quit
s	to save the game for later

When **backgammon** prompts with 'Move:', type

p	to print the board
q	to quit
s	to save the game

or a *move,* which is a sequence of

s-f	move from s to **f**
s/r	move one man on **s** the roll **r** separated by commas or spaces and ending with a newline. Available abbreviations are

s-f1-f2 means **s-f1,f1-f2**

s/r1r2 means **s/r1,s/r2**

Use **b** for bar and **h** for home, or **0** or **25** as appropriate.

FILES

/usr/games/teachgammon	rules and tutorial
/etc/termcap	terminal capabilities

BUGS

backgammon's strategy needs much work.

NAME

banner – print large banner on printer

SYNOPSIS

/usr/games/banner [–w*n*] message . . .

DESCRIPTION

banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If –**w** is given, the output is reduced from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably ‘<’, ‘>’, ‘[’, ‘]’, ‘\’, ‘^’, ‘_’, ‘{’, ‘}’, ‘|’, and ‘~’. Also, the characters ‘"’, ‘'’, and ‘&’ are funny looking (but in a useful way.)

The –**w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

NAME

battlestar – a tropical adventure game

SYNOPSIS

battlestar [**–r**]

DESCRIPTION

battlestar is an adventure game in the classic style. However, it is slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

OPTIONS

–r Recover a saved game.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

Three He made and gave them to His daughters,
Beautiful nymphs, the goddesses of the waters.
One to bring good luck and simple feats of wonder,
Two to wash the lands and churn the waves asunder,
Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word '**su**', could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

USAGE

Sample Commands

```
take     ---   take an object
drop     ---   drop an object
wear     ---   wear an object you are holding
draw     ---   carry an object you are wearing
puton    ---   take an object and wear it
take off ---   draw an object and drop it
throw  <object> <direction>
!        <shell esc>
```

Implied Objects

```
>-: take watermelon
watermelon:
Taken.
>-: eat
watermelon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
Taken.
broadsword:
Taken.
apple:
Taken.
knife:
Dropped.
```

broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying '**take knife**' and then '**drop**' will drop the knife you just took.

Score and Inven

The two commands **score** and **inven** will print out your current status in the game.

Saving a Game

The command **save** will save your game in a file called **Bstar**. You can recover a saved game by using the **–r** option when you start up the game.

Directions

The compass directions N, S, E, and W can be used if you have a compass. If you do not have a compass, you will have to say **R**, **L**, **A**, or **B**, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

BUGS

Countless.

NAME

bcd, ppt – convert to antique media

SYNOPSIS

/usr/games/bcd *text*

/usr/games/ppt

DESCRIPTION

bcd converts the literal *text* into a form familiar to old-timers.

ppt converts the standard input into yet another form.

SEE ALSO

dd(1)

NAME

bdemos – demonstrate Sun Monochrome Bitmap Display

SYNOPSIS

/usr/demo/bballs
/usr/demo/bbounce
/usr/demo/bdemos
/usr/demo/bjump
/usr/demo/bphoto *file*
/usr/demo/brotcube

DESCRIPTION

Bdemos is a collection of simple demonstration programs for the Sun Monochrome Bitmap Display. Each program is briefly described below. Unless otherwise noted, each program should be terminated by typing the appropriate key (usually DELETE or ^C) to generate an interrupt signal.

bballs colliding balls demo

bbounce bouncing square demo

bdemos a collection of demos

This program has a menu for selection of several different demos. After typing a key to select a particular demo, the user may type ^C to get back the menu. Type 'q' to quit.

bjump simulated jump to hyperspace

bphoto *file* dither monochrome image *file* to bitmap display

Image files suitable for display by this program are in */usr/demo/bwpix*.

brotcube black and white spinning cube

FILES

/usr/demo/bwpix

SEE ALSO

bsuncube(6), **draw**(6)

NAME

bj – the game of black jack

SYNOPSIS

/usr/games/bj

DESCRIPTION

bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

> The bet is $2 every hand.
>
> A player "natural" (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a "push" (no money exchange).
>
> If the dealer has an ace up, the player is allowed to make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.
>
> If the player is dealt two cards of the same value, he is allowed to "double". He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)
>
> If a dealt hand has a total of ten or eleven, the player may "double down". He may double the bet ($2 to $4) and receive exactly one more card on that hand.
>
> Under normal play, the player may "hit" (draw a card) as long as his total is not over twenty-one. If the player "busts" (goes over twenty-one), the dealer wins the bet.
>
> When the player "stands" (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.
>
> If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by y followed by a new-line for "yes", or just new-line for "no".

> **?** (this means, "do you want a hit?")
> **Insurance?**
> **Double down?**

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, hit the interrupt key (CTRL–C) and the action and standing will be printed.

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

boggletool – play a game of boggle

SYNOPSIS

/usr/games/boggletool [*number*] [+[+]] [16-character *string*]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

boggletool allows you to play the game of Boggle (TM Parker Bros.) against the computer. The *number* argument specifies the time limit in minutes (the default is 3 minutes). If a 16 character long string is placed on the command line, it is interpreted as a Boggle board: the first four letters form the top row, the next four letters the second row, etc. If no letters are specified, a board is randomly rolled by the computer from a set of Boggle cubes. The +[+] argument is explained below under **Advanced Play .**

PLAYING THE GAME

Rules of the Game

The object of Boggle is to find as many words as possible in a 4 by 4 grid of letters within a certain time limit. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. Normally, no letter in the grid may be used more than once in a word (see **Advanced Play** for exceptions).

Playing the Game

When invoked, boggletool displays a grid of letters and an hourglass. To enter words, simply type in lower case letters to spell the word you want. Use any whitespace (SPACE, TAB, or NEWLINE) to finish a word. To correct mistakes you make, use BACKSPACE or DEL to delete the last character, or use CTRL-U to delete an entire word. **boggletool** verifies that words you enter are both in the grid and are valid English words. If you type in a character which would form a word which is not in the grid, the display will flash and the character you typed will not be echoed. When you type any whitespace to end the current word, **boggletool** will verify that the word is three or more letters long and that it appears in the dictionary. If the word you typed is illegal for either reason, the display will flash and you will have to either erase the word or change it. If you try to enter a valid word which you have already entered, the display will flash and the previous occurrence of the word will be highlighted. Again, you will have to erase the word before continuing. As you enter words, the "sand" in the hourglass will fall. At the end of the time limit, the display will flash and you will no longer be allowed to enter words. After a moment, the computer will display two lists of words: the words you found, and other words which also appear in the grid. To play another game, just type any capital letter (or use the pop-up menu).

Using the Menu

The pop-up menu is invoked by pressing the RIGHT mouse button. There are four items in it, and they work as follows.

Restart Game
: Create a new boggletool a new board, reset the timer, and allow you to start from scratch.

Restart Timer
: Allows you to cheat by reseting the hourglass timer to zero.

Give Up
: End the game and print the results immediately.

Quit
: Allows you to quit running the boggletool program. A prompt appears asking you to confirm the quit; when it does, click the LEFT mouse button to quit or the RIGHT mouse button to abort the quit.

Advanced Play

There are two options for advanced players. If a single + appears on the command line, letters in the grid may be reused. If two +'s are on the command line, letters may also be considered adjacent to themselves as well as to their neighbors. Although it is far easier to find words with these two options, there are also many more possible words in the grid and it is therefore difficult to find them all.

FILES

/usr/games/boggledict dictionary file for computer's words

NAME

brotcube – rotate a simple cube

SYNOPSIS

/usr/demo/brotcube

DESCRIPTION

brotcube rotates a skeletal outline of a cube consisting of 14 vectors. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. Each rotation consists of 100 views.

This program gives an indication of the performance of the SunCore Graphics Package.

Type **q** to exit the program.

NAME

bsuncube – view 3-D Sun logo

SYNOPSIS

/usr/demo/bsuncube

DESCRIPTION

bsuncube allows the user to view a cube from various positions with hidden faces removed. The faces of the cube consist of the Sun logo. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display.

The program operates in two modes: **DisplayObject** mode and **SelectView** mode. The program starts in **DisplayObject** mode:

> **DisplayObject**: The cube is displayed in 3-D perspective with hidden faces removed. Type **q** while in this mode to exit the program. Press RIGHT mouse button to switch to **SelectView** mode.
>
> **SelectView**: Schematic projections of the outline of the cube are shown and the mouse is used to select a viewing position. Use LEFT mouse button to set *x* and MIDDLE mouse button to set *y* in the *Front View*. Use MIDDLE mouse button to set *z* in the *Top View*. Press RIGHT mouse button to switch to **DisplayObject** mode.

The view shown in **DisplayObject** mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the cube and that the positive *y* axis on the screen is the projection of the positive *y* axis in 3-D cube coordinates.

NAME

buttontest – demonstration and testing program for SunButtons

SYNOPSIS

/usr/demo/BUTTONBOX/buttontest

DESCRIPTION

buttontest displays a window with thirty two buttons, corresponding to those on SunButtons. To determine if the button box has been set up correctly, select the **Diagnostic** button on the panel. If the button box is correctly interfaced, **buttonbox OK** is displayed, and pressing a button on the box highlights a button on the screen. If **No Response from Buttonbox** is displayed, repeat the button box install procedure.

NAME

canfield, canfieldtool, cfscores – Canfield solitaire card game

SYNOPSIS

/usr/games/canfield [**–ac**]

/usr/games/canfieldtool [**–ac**]

/usr/games/cfscores [**–ac**] [*username*]

AVAILABILITY

These games are available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

canfield can be played on any terminal. **canfieldtool** is the SunView version with attractive graphics.

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In **canfield,** tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types **ht** for his move. Foundation base cards are also automatically moved to the foundation when they become available.

Canfieldtool

Once you understand the rules, **canfieldtool** is self-explanatory.

Canfield

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (The initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

OPTIONS

a Print out **canfield** accounts for all users that have played the game since the database was set up.

c Maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase the chances of winning.

With no arguments, **cfscores** prints out the current status of your canfield account. If *username* is specified, it prints out the status of their account.

FILES

/usr/games/canfield the game itself
/usr/games/lib/cfscores the database of scores

BUGS

It is impossible to cheat.

NAME

cdplayer – CD-ROM audio demo program

SYNOPSIS

cdplayer [-**d** *device*] [*sunview options*]

AVAILABILITY

This demo is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

cdplayer demonstrate the CD quality audio capability of the CD-ROM drive. It is a SunView program and plays any Audio Compact Discs. There are four panels in the window. The top panel displays the all the available tracks on the CD. The user can select the any tracks by clicking it with the left mouse button. The second panel contains the play, pause, stop and eject button. The third panel display the CD music address and track number. The bottom panel contains the volume control slider and close button.

Refer to the CD-ROM hardware documentation for connecting the speakers or head-phones to the drive.

OPTIONS

–d *device* Use *device* as the CD-ROM device, rather than **/dev/rsr0** the default CD-ROM device.

FILES

/dev/rsr0 CD-ROM raw file

SEE ALSO

sr(4)

NAME

chess – the game of chess

SYNOPSIS

/usr/games/chess

AVAILABILITY

This game is available for Sun-3 and Sun-4 systems with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; '**o-o**' and '**o-o-o**' specify castling. To play black, type '**first**'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

DIAGNOSTICS

The most cryptic diagnostic is '**eh?**' which means that the input was syntactically incorrect.

FILES

/usr/games/lib/chess.book
book of opening moves

BUGS

Pawns may be promoted only to queens.

NAME

chesstool – window-based front-end to chess program

SYNOPSIS

/usr/games/chesstool [*chess_program*]

AVAILABILITY

This game is available for Sun-3 and Sun-4 systems, with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

chesstool is a window-based front-end to the **chess**(6) program. Used without options, **chesstool** uses **/usr/games/chess**; you can designate any alternate program which uses the same command syntax as **chess**(6) with the *chess_program* argument.

When **chesstool** starts up, it displays a large window with three subwindows. The first subwindow displays messages '**Illegal move**', for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Items in the subwindow may be selected with either the left or middle mouse buttons. These options are:

Last Play Show the last play made.

Undo Undo your last move and the machine's response.

Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.

Flash Flash when the machine has completed its move.

When this command is selected, a check mark will appear next to the word **Flash**. In flash mode, if **chesstool** is open, the piece moved by the system on its play will flash until you make your move. If **chesstool** is iconic, the entire icon will flash when the machine has made its move. Thus you can "Close" **chesstool** and be alerted when it's your turn to move. To turn flash mode off, select flash again.

Machine White Start a new game with the machine playing white.

Human White Start a new game with the machine playing black.

Quit Exit from **chesstool**.

There are two moves which are special: castling and capturing a pawn *en*passant. To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en*passant, move the pawn to the square occupied by the opposing pawn which will be captured.

SEE ALSO

chess(6)

NAME

ching – the book of changes and other cookies

SYNOPSIS

/usr/games/ching [*hexagram*]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight (—–) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each *hexagram* consists of two major sections. The **Judgement** relates specifically to the matter at hand (For instance, "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the *hexagram* and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines has the value six or nine, it is a moving line; for any such line there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second *hexagram* (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting *hexagram* will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, this oracle simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process ID and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try **fortune**(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

colordemos – demonstrate Sun Color Graphics Display

SYNOPSIS

/usr/demo/cballs
/usr/demo/cdraw
/usr/demo/cphoto *file*
/usr/demo/cpipes
/usr/demo/cshowmap *file*
/usr/demo/csnow
/usr/demo/csuncube
/usr/demo/csunlogo
/usr/demo/cvlsi

DESCRIPTION

colordemos is a collection of simple demonstration programs for the Sun Color Graphics Display. Each program is briefly described below. To exit each program, send an interrupt signal by typing the appropriate key (usually CTRL-C).

cballs	Colliding balls on color display.
cdraw	Draw on the color display (see **draw**(6) for an explanation of how to use **cdraw**).
cphoto *file*	Display dithered color file on color display. Files suitable for display are in **/usr/demo/colorpix.**
cpipes	Colliding pipes on color display.
cshowmap *file*	Display maps. Files suitable for display are in **/usr/demo/segments**.
csnow	Color kaleidoscope.
csuncube	Multicolored Sun logo.
csunlogo	Shaded Sun logo.
cvlsi	Color VLSI layout demo.

FILES

/usr/demo/colorpix
/usr/demo/segments

NAME

craps – the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a ''bankroll'' of $2,000.

The program prompts with:

bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;
any other number	is the *point*, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

marker?

A **yes** (or **y**) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

Repay marker?

A reply of **yes** (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed will be *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than **yes** is considered to be a **no** (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DELETE character or CTRL–D The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**–eqr**] *name* . . .

DESCRIPTION

cribbage plays the card game cribbage, with **cribbage** playing one hand and the user the other. **cribbage** initially asks the user if the rules of the game are needed – if so, **cribbage** displays the appropriate section from *According to Hoyle* with **more**(1).

OPTIONS

–e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

–q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

–r Instead of asking the player to cut the deck, **cribbage** will randomly cut the deck.

PLAYING CRIBBAGE

cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of '**s**' results in a short game, any other response plays a long game.

At the start of the first game, **cribbage** asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, **cribbage** first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, **cribbage** cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. **cribbage** keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. **cribbage** requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *rank*s may be specified as one of **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **t**, **j**, **q**, and **k**, or alternatively, one of **ace**, **two**, **three**, **four**, **five**, **six**, **seven**, **eight**, **nine**, **ten**, **jack**, **queen**, and **king**. *Suit*s may be specified as **s**, **h**, **d**, and **c**, or alternatively as **spades**, **hearts**, **diamonds**, and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h**, **4d**, **5c**, **6h**, **jc**, **kd** and you wanted to discard the king of diamonds, you could type any of **k**, **king**, **kd**, **k d**, **k of d**, **king d**, **king of d**, **k diamonds**, **k of diamonds**, **king diamonds**, or **king of diamonds**,

FILES

/usr/games/cribbage

SEE ALSO

more(1)

NAME

dialtest – demonstration and testing program for SunDials

SYNOPSIS

/usr/demo/DIALBOX/dialtest

DESCRIPTION

dialtest displays a window with eight dials, corresponding to those on SunDials. To determine if the dialbox has been set up correctly, select the **Diagnostic** button on the panel. If the dialbox is correctly interfaced, **Dialbox OK** is displayed, and turning a dial on the box turn a dial on the screen. If **No Response from Dialbox** is displayed, repeat the dialbox install procedure.

NAME

draw, bdraw, cdraw – interactive graphics drawing

SYNOPSIS

/usr/demo/bdraw
/usr/demo/cdraw

DESCRIPTION

The *draw* programs are menu-driven programs which use the mouse, keyboard, bitmap display and optionally the color display to draw objects, drag them around, save them on disk, and so on. **bdraw** is the draw program for the black and white display and **cdraw** is the program for driving the color display.

The main menu items are selected by moving the mouse cursor and pressing the left mouse button. To redraw the display, point at the left edge of the main menu box and press the left button. The main menu items are:

New Seg xlate
Open a new translatable segment. A segment is a collection of attributes and primitives (lines, text, polygons, etc.). A translatable segment may subsequently be positioned.

New Seg xform
Open a new transformable segment. A transformable segment may subsequently be rotated, scaled, or positioned.

Delete Seg To delete a segment, point at any primitive in the segment and press the left button.

Lines To add line primitives to the currently open segment, position cursor, press the left button, ... press right button to quit.

Polygon To add a polygon primitive to the currently open segment, position the cursor, press the left button, ... press the right button to terminate the boundary definition. Polygons are filled with the current fill attribute.

Raster To add a raster primitive to the currently open segment, position the cursor, press the left button to reposition the box, adjust the box by moving the mouse, press the right button to create the raster primitive comprising the boxed bitmap. A 'rasterfile' is also created on disk for hardcopy purposes (see */usr/include/rasterfile.h*). This 'rasterfile' file may be spooled to a Versatec printer/plotter for hardcopy after exiting from the draw program. The command to do this is **lpr –v rasterfile.**

Text To add a text primitive to the currently open segment, position cursor, press left button, type the text string at the keyboard (back space works), hit return. Text is drawn with the current text attributes.

Marker To add marker primitives to the currently open segment, position cursor, press the left button to place marker, ... press the right button to quit.

Position To position a segment, point at any primitive in the segment, press left button, position the segment, press right button to quit.

Rotate To rotate a transformable segment, point at any primitive in the segment, press left button, move mouse to rotate, press right button to quit.

Scale To scale a transformable segment, point at any primitive in the segment, press the left button, move mouse to scale in x or y, press right button to quit.

Attributes This item brings up the attribute menu. To select an attribute such as text font, region fill texture (color), linestyle, or line width, point at the item and press the left button. Point at the left edge of the menu box to quit.

Save Seg To save a segment on a disk file, point at the segment, press the left button, type the disk file name, hit return.

Restore Seg
To restore a previously saved segment from disk, type file name, hit return.

Exit Exit the draw program.

BUGS

Rasters and raster text do not scale or rotate. If segments completely overlap, only the last one drawn may be picked by pointing with the mouse. This also applies to the menu segments! Therefore, don't cover them up with polygons. If aborted with your interrupt character, you must give the 'reset' command to turn keyboard echo back on and to reset -cbreak. Therefore, use the Exit item in the main menu to exit the program.

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

/usr/games/factor [*number*]

/usr/games/primes [*number*]

DESCRIPTION

factor reads lines from its standard input. If it reads a positive number, **factor** will factor the number and print its prime factors, printing each one the proper number of times. **factor** exits when it reads zero, a negative number, or something other than a number. If a *number* is given, **factor** will factor the number, print its prime factors, and exit.

primes reads a number from the standard input and prints all primes larger than the given number and smaller than 2^{32} (about 4.3×10^{9}). If a *number* is given, **primes** will use that number rather than reading one from the standard input.

DIAGNOSTICS

Ouch. Input out of range or for garbage input.

NAME

fish – play ''Go Fish''

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of "Go Fish", a children's card game. The object is to accumulate "books" of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the "pool" of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **j**, **q**, or **k** when asked. Hitting a RETURN character gives you information about the size of my hand and the pool, and tells you about my books. Saying '**p**' as a first guess puts you into "pro" level; the default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [**–alsw**] [*filename*]

DESCRIPTION

fortune with no arguments prints out a random adage. The flags mean:

- **–a** Choose from either list of adages.
- **–l** Long messages only.
- **–s** Short messages only.
- **–w** Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

FILES

/usr/games/lib/fortunes.dat

NAME

gaintool – audio control panel

SYNOPSIS

gaintool

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

gaintool is a SunView demonstration program that controls various characteristics of the SPARCstation 1 audio device, see **audio**(4S). Operations performed by **gaintool** affect all audio programs; for instance, adjusting the **Play Volume** instantly changes the output gain, regardless of which program is playing. **gaintool** also detects audio state changes made by other programs, and updates its display accordingly, keeping **gaintool** in sync with the current device configuration.

gaintool demonstrates an important principle involved in the integration of audio in the desktop environment: by enabling global control of important characteristics, it is not necessary for every application to provide an interface for these parameters. For instance, since audio output may be paused from the control panel, it is not strictly necessary that output applications display a **Pause** button of their own. However, such applications may detect that audio output has been paused, and take appropriate action.

Control Panel

Play Volume
This slider adjusts the output volume. Volume levels between 0 and 100 may be selected, where 0 represents infinite attenuation and 100 is maximum gain.

Record Volume
This slider adjusts the recording gain level in the range 0 to 100.

Monitor Volume
This slider adjusts the monitor gain level in the range 0 to 100. Monitor gain controls the amount of audio input signal that is fed through to the output port. For instance, if an audio source (such as a radio or CD-player) is connected directly to the input port, the input signal may be monitored through either the built-in speaker or the headphone jack.

Output This selector switches the audio output port between the built-in speaker and the external headphone jack.

Pause Play
This button may be used to suspend and resume audio output. If audio output is in progress when **Pause** is clicked, it is stopped immediately and subsequent output data remains queued. The button then switches to a **Resume** button that, when clicked, resumes audio output at the point that it was suspended.

If no process has the device open for output when **Pause** is clicked, **gaintool** holds the device open itself, thereby denying other processes output access. Audio programs that simply open and write to the audio device will typically be suspended when they attempt to open the device. Programs that asynchronously poll the device will discover that it is "busy" and may take appropriate action.

Audio Device Status Panel

Pressing the **PROPS** (L3) key brings up a status panel that shows the current state with the its display accordingly, audio applications. Selecting "Done" from the panel menu (or pressing the (L7) key) removes the panel.

Ordinarily, the device status is updated only when a **SIGPOLL** signal is delivered to **gaintool** (see **audio**(4S)). Because of this, the **Active** and **Samples** indicators are not necessarily kept up-to-date. However, when the mouse is positioned over the panel, status is continually updated.

SEE ALSO

audio(4S), **soundtool**(6)

BUGS

Record Volume should be controlled by a separate panel that also provides automatic gain level adjustment capabilities.

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

gammontool – play a game of backgammon

SYNOPSIS

/usr/games/gammontool [*path*]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

gammontool paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with **gammontool**.

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

The Initial Roll

To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

Making Your Move

When it is your turn, '**Your**move' appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

Doubling

To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the **Accept Double** or **Refuse Double** buttons. The **Forfeit** button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

Other Buttons

If you want to change your move before you have finished it, use the **Redo Move** or **Redo Entire Move** buttons in the option window. **Redo Entire Move** replaces all of the pieces you have moved so that you can redo them all. **Redo Move** only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The **Show Last Move** button allows you to see the last move again.

Leaving the Game

If you want to quit playing backgammon, use the **Quit** button. If you want to forfeit the game, use the **Forfeit** button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click the **New Game** button. To change the color of your pieces, click the mouse button while pointing at either the **White** or **Black** checkboxes. You may change colors at any time, even in the middle of a game. Changing colors in the middle of a game does not mean that you trade places with the computer; your pieces stay where they are, but they are repainted with the new color. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice are always displayed on the left half of the board.

Log File

If a there is a **gammonlog** file your home directory, **gammontool** keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

FILES

~/gammonlog log of games played

/usr/games/lib/gammonscores
log of wins and losses

BUGS

The default strategy used by the computer is very poor.

If a single move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without touching down in the middle) it is unpredictable where the program will make the piece touch down. This may be important if there is a blot on one of these middle points. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

NAME

gp_demos, flight, rotobj – demonstration programs for the Graphics Processor

SYNOPSIS

/usr/demo/flight

/usr/demo/rotobj [*object*]

AVAILABILITY

These demos are available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

These demos only run in windows running on a Graphics Processor surface.

Flight

flight is a mouse-driven flight simulator.

Interactive Commands

Middle-Button Restart the program.

Right-Button Increase speed.

Left-Button Decrease speed.

Move-Mouse-Forward
The airplane dives.

Move-Mouse-Backward
The airplane climbs.

Move-Mouse-Left/Right
The airplane banks.

Left/Right-With-Right-Button
The airplane rolls without banking.

Rotobj

rotobj rotates an *object*. Object files are located in **/usr/demo/DATA** and have the suffix **.vecs**.

FILES

/usr/demoDATA

SEE ALSO

graphics_demos(6)

NAME

graphics_demos, bouncedemo, framedemo, jumpdemo, spheresdemo, – graphics demonstration programs

SYNOPSIS

/usr/demo/bouncedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/framedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/jumpdemo [**–c**] [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/spheresdemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

AVAILABILITY

These demos are available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

bouncedemo

bouncedemo displays a bouncing square.

framedemo

framedemo

displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). **framedemo** looks for the frames in the files **frame.1** through **frame.n** in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory **/usr/demo/globeframes/***.

Interactive Commands

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

f Remove 1/20th of a second from the interval.

F Remove one second from the interval. **Ff** makes the interval as small as possible.

s Add 1/20th of a second.

S Add one second.

jumpdemo

jumpdemo simulates the famous **Star Wars** jump to light-speed-sequence using vector drawing. Colored stars are drawn on color surfaces.

spheresdemo

spheresdemo computes a random collection of shaded spheres. Colored spheres are drawn on color surfaces.

OPTIONS

–c Rotate the color map to produce a sparkling effect.

–d *surface*
Run the demo on a surface other than the window or system console, for instance:

bouncedemo –d /dev/cgone0

–n*x* Draw *x* items, or repeat a sequence *x* times.

–r Retain the window. This allows the image to reappear when uncovered instead of restarting the demo.

–q Quick exit. Useful for running several demos from within a shell script.

NAME

hack – replacement for rogue

SYNOPSIS

hack [–d *hackdir*] [**–s all** | *player* ...]

DESCRIPTION

hack is a display-oriented dungeons & dragons type game. Both display and command structure resemble *rogue*, although **hack** has twice as many monster types and requires three times as much memory.

Normally **hack** looks in **/usr/games/lib/hackdir** for the files listed below; this directory can be changed with the **–d** option. The –s option permits you to search the player record. Given the keyword **all**, **hack** lists all players; given the login name of a player, it lists all scores of that player.

FILES

record	top 100 list (start with an empty file)
news	changes or bugs (start with no news file)
data	information about objects and monsters
help	introductory information (no doubt outdated)
hh	compacted version of help
perm	empty file used for locking
rumors	texts for fortune cookies

NAME

hangman – computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In **hangman,** the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words on-line word list

NAME

hunt – a multiplayer multiterminal game

SYNOPSIS

/usr/games/hunt[**–m**] [*hostname*] [**–l** *name*]

DESCRIPTION

The object of the game **hunt** is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of **hunt**. The more players you kill before you die, the better your score is.

hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname*argument.

hunt only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that do not fit in the status area.

hunt uses the same keys to move as **vi** does, for instance, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (for instance, HJKL).

Other commands are:

f	Fire (in the direction you're facing) (Takes 1 charge)
g	Throw grenade (in the direction you're facing) (Takes 9 charges)
F	Throw satchel charge (Takes 25 charges)
G	Throw bomb (Takes 49 charges)
o	Throw small slime bomb (Takes 15 charges)
O	Throw big slime bomb (Takes 30 charges)
s	Scan (where other players are) (Takes 1 charge)
c	Cloak (where you are) (Takes 1 charge)
^L	Redraw screen
q	Quit

Knowing what the symbols on the screen often helps:

–|+ Walls

/\h|288u+288uDiagonal (deflecting) walls

#	Doors (dispersion walls)
;	Small mine
g	Large mine
:	Shot
o	Grenade
O	Satchel charge
@	Bomb
s	Small slime bomb
$	Big slime bomb
><^v	You facing right, left, up, or down

} { i ! Other players facing right, left, up, or down

* Explosion

```
 \|/
-*E-   Grenade and large mine explosion
 /|\
```

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

You can only fire in the direction you are facing.
You can only fire three shots in a row, then the gun must cool.
A shot only affects the square it hits.
Shots and grenades move 5 times faster than you do.
To stab someone,
    you must face that player and move at them.
Stabbing does 3 points worth of damage and shooting does 5 points.
You start with 15 charges and get 5 more for every new player.
A grenade affects the nine squares centered about the square it hits.
A satchel affects the twenty-five squares centered about the square it hits.
A bomb affects the forty-nine squares centered about the square it hits.
One small mine and one large mine is placed in the maze for every new player.
A mine has a 5% probability of tripping when you walk directly at it;
    50% when going sideways on to it; 95% when backing up on to it.
Tripping a mine costs you 5 points or 10 points respectively.
Defusing a mine is worth 1 charge or 9 charges respectively.
You cannot see behind you.
Scanning lasts for (20 times the number of players) turns.
    Scanning takes 1 ammo charge, so do not waste all your charges scanning.
You get 2 more damage capacity points and 2 damage points taken away
    whenever you kill someone.
Maximum typeahead is 5 characters.
A shot destroys normal (for instance, non-diagonal, non-door) walls.
Diagonal walls deflect shots and change orientation.
Doors disperse shots in random directions (up, down, left, right).
Diagonal walls and doors cannot be destroyed by direct shots but may
    be destroyed by an adjacent grenade explosion.
Walls regenerate, reappearing in the order they were destroyed.
    One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.

ENVIRONMENT

The environment variable **HUNT** is checked to get the player name. If you do not have this variable set, **hunt** will ask you what name you want to play under. You may also set up a single character keyboard map, but then you have to enumerate the options. For example:

setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"

sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**.

The *mapkey* option must be last.

It is a boring game if you are the only one playing.

OPTIONS

−m You enter the game as a monitor (you can see the action but you cannot play).

−l *name* Enter the game as player *name*.

FILES

/usr/games/lib/hunt.driver game coordinator

LIMITATIONS

hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a **hunt** game executing. A limit of three players per host and nine players total is enforced by **hunt**.

BUGS

To keep up the pace, not everything is as realistic as possible.

NAME

life – John Conway's game of life

SYNOPSIS

/usr/games/life

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

life is a program that plays John Conway's game of life. It only runs under **sunview**(1).

When invoked, **life** will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. **life** keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occurred. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces disappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** Exits the tool. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

SEE ALSO

sunview(1)

NAME

mille – play Mille Bornes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bornes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

- P — Pick a card from the deck. This card is placed in the 'P' slot in your hand.
- D — Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a carriage-return or space. The carriage-return or space is required to allow recovery from typos which can be very expensive, like discarding safeties.
- U — Use a card. The card is again indicated by its number, followed by a carriage-return or space.
- O — Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
- Q — Quit the game. This will ask for confirmation, just to be sure. Hitting DELETE (or RUBOUT) is equivalent.
- S — Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a RETURN character without a name, the save will be terminated and the game resumed.
- R — Redraw the screen from scratch. The command ^L (CTRL–L) will also work.
- W — Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save. (The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3V)

CARDS

Here is some useful information. The number in brackets after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas [2]	Gasoline [6]	Extra Tank [1]
Flat Tire [2]	Spare Tire [6]	Puncture Proof [1]
Accident [2]	Repairs [6]	Driving Ace [1]
Stop [4]	Go [14]	Right of Way [1]
Speed Limit [3]	End of Limit [6]	

25 – [10], 50 – [10], 75 – [10], 100 – [12], 200 – [4]

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. With the exception of the *speed limit* card, they can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA (unlabeled)**: This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand*. A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

Coup Fouré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bornes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.

Each Safety: 100 points for each safety in the Safety area.

All 4 Safeties: 300 points if all four safeties are played.

Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.

Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.

Delayed Action: 300 points bonus for finishing after the deck was exhausted.

Extension: 200 points bonus for completing a 1000 mile trip.

Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop [*filename*]

DESCRIPTION

monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", that is, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, for instance a name, place or person, you can type **?** to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

USAGE

Commands

quit: Quit game. This allows you to quit the game. It asks you if you are sure.

print Print board. This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings:
List your own holdings, that is, money, get-out-of-jail-free cards, and property.

holdings:
Holdings list. Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type **done**.

shell: Shell escape. Escape to a shell. When the shell dies, the program continues where you left off.

mortgage:
Mortgage property. Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
Unmortgage property. Unmortgage mortgaged property.

buy: Buy houses. Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: Sell houses. Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to **buy**

card: Card for jail. Use a get-out-of-jail-free card to get out of jail. If you are not in jail, or you do not have one, it tells you so.

pay: Pay for jail. Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you are not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: Save game. Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the **monop** command, or by using the **restore** command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore:
Restore game. Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the RETURN key instead of a command, it is the same as typing *roll*.

FILES

/usr/games/lib/cards.pck chance and community chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

moo – guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A ''cow'' is a correct digit in an incorrect position. A ''bull'' is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

play – play audio files

SYNOPSIS

play [**–i**] [**–V**] [**–d** *dev*] [**–v** *vol*] [*filename* ...]

AVAILABILITY

This command is only available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

play copies the named audio files to the audio device. Audio files named on the command line are played sequentially. If no filenames are present, the standard input stream is played. The special filename '–' may be used to read the standard input stream instead of a file.

The input files (including the standard input) must contain a valid audio file header. The encoding information in this header is matched against the capabilities of the audio device and, if the data formats are incompatible, an error message is printed and the file is skipped.

Minor deviations in sampling frequency (those less than 1%) are ordinarily ignored. This allows, for instance, data sampled at 8012 Hz to be played on an audio device that only supports 8000 Hz. If the –V option is specified, such deviations are flagged with warning messages.

OPTIONS

–i Print an error message and exit immediately if the audio device is unavailable (that is, another process currently has write access). **play** will ordinarily wait until it can obtain access to the device.

–V Verbose. Print messages to the standard error while waiting for access to the audio device or when sample rate deviations are detected.

–d *dev* Specify an alternate audio device to which output should be directed. If the **–d** option is not specified, **/dev/audio** is the default audio device.

–v *vol* Set the output volume to *vol* before playing begins. *vol* is an integer value between 0 and 100, inclusive. If this argument is not specified, the output volume remains at the level most recently set by any process.

–? Help. Print a command line usage message.

SEE ALSO

record(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

primes – print all primes larger than some given number

SYNOPSIS

/usr/games/primes [*number*]

DESCRIPTION

primes reads a number from the standard input and prints all primes larger than the given number. If *number* is given as an argument, it uses that number rather than reading one from the standard input.

BUGS

It obviously cannot print *all* primes larger than some given number. It will not behave very sensibly when it overflows an **int**.

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [*–ifilename*] [**–t**] [*category1 category2*]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, **quiz** gives instructions and lists the available categories.

quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

The **–t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **–i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category = alternate | category '|' alternate
alternate = empty | alternate primary
primary  = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with **sh**(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a | ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use **termcap**, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

NAME

random – select lines randomly from a file

SYNOPSIS

/usr/games/random [**–er**] [*divisor*]

DESCRIPTION

random acts as a text filter, randomly selecting lines from its standard input to write to the standard output. The probability that a given line is selected is normally 1/2; if a *divisor* is specified, it is treated as a floating-point number, and the probability is 1/*divisor* instead.

OPTIONS

–e Don't read the standard input or write to the standard output. Instead, exit with a random exit status between 0 and 1, or between 0 and *divisor*-1 if *divisor* is specified.

–r Don't buffer the output. If **–r** is not used, output is buffered in blocks, or line-buffered if the standard output is a terminal.

NAME

raw2audio – convert raw audio data to audio file format

SYNOPSIS

raw2audio [**–f**] [**–c** *chan*] [**–e** *enc*] [**–i** *info*] [**–o** *cnt*] [**–p** *bits*] [**–s** *rate*] [*filename* ...]

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

raw2audio adds an audio file header to the named raw data files. The encoding information in this header is taken from the command line options.

If no filenames are specified, **raw2audio** reads raw data from the standard input stream and writes an audio file to the standard output. If a target file is a symbolic link, the underlying file will be rewritten.

OPTIONS

–f Force. If an input file already contains an audio file header, **raw2audio** ordinarily prints a warning message and skips the file. If the **–f** flag is specified, the old file header, including the 'information' field, is replaced.

–c *chan* Specify the number of interleaved audio channels in each sample frame. If not specified, a single channel is assumed.

–e *enc* Specify the encoding type. *enc* may be one of the following: **ULAW**, **LINEAR**, or **FLOAT**, corresponding to μ-law, integer PCM, and IEEE floating-point formats, respectively. If not specified, μ-law encoding is assumed.

–i *info* Specify the 'information' field of the output file header.

–o *cnt* Specify the number of bytes to skip in the audio data stream. This option may be used, for instance, to extract audio data from files containing unrecognizable file headers.

–s *rate* Specify the sample rate frequency, in Hz. If not specified, the sample rate defaults to 8000 Hz.

–p *bits* Specify the sound unit size, in bits. If not specified, the precision defaults to 8 bits.

–? Help. Print a command line usage message.

SEE ALSO

play(6), **record**(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

record – record an audio file

SYNOPSIS

record [**–a**] [**–f**] [**–d** *dev*] [**–i** *info*] [**–t** *time*] [–v *vol*] [*filename*]

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

record copies audio data from the audio device to a named audio file. The output file will be prefixed by an audio file header. The encoding information in this header is taken from the configuration of the audio device. If no filename is present, or if the special filename '–' is specified, output is directed to the standard output stream.

Recording begins immediately and continues until a **SIGINT** signal (CTRL-C) is received. If the **–t** option is specified, *record* stops when the specified quantity of data has been recorded.

If the audio device is unavailable (that is, another process currently has read access), **record** prints an error message and exits immediately.

OPTIONS

–a Append the data on the end of the named audio file. The audio encoding of the file must match the audio device configuration.

–f Force. When the **–a** flag is specified, the sample rate of the audio file must match the device configuration. If the **–f** flag is also specified, sample rate differences are ignored, with a warning message printed to the standard error.

–d *dev* Specify an alternate audio device from which input should be taken. If the **–d** option is not specified, **/dev/audio** is used as the default audio device.

–i *info* The 'information' field of the output file header is set to the string specified by the *info* argument. This option may not be specified in conjunction with the **–a** argument.

–t *time* The *time* argument specifies the maximum length of time to record. Time may be specified as a floating-point value, indicating the number of seconds, or in the form: *hh*:*mm*:*ss*.**dd**, where hour and minute specifications are optional.

–v *vol* Specify the recording gain. *vol* is an integer value between 0 and 100, inclusive. If this argument is not specified, the input volume will remain at the level most recently set by any process.

–? *Help*: Print a command line usage message.

SEE ALSO

play(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

robots – fight off villainous robots

SYNOPSIS

/usr/games/robots [**–sjta**] [*scorefile*]

DESCRIPTION

robots pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by a '+', the junk heaps from their collisions by a '*', and you (the good guy) by a '@'.

The commands are:

h move one square left

l move one square right

k move one square up

j move one square down

y move one square up and left

u move one square up and right

b move one square down and left

n move one square down and right

. (also space) do nothing for one turn

HJKLBNYU
run as far as possible in the given direction

> do nothing for as long as possible

t teleport to a random location

w wait until you die or they all do

q quit

^L redraw the screen

All commands can be preceded by a count.

If you use the '**w**' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with '**w**' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

OPTIONS

–s Do not play, just show the score file.

–j Jump, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.

−t Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.

−a Advance into the higher levels directly, skipping the lower, easier levels.

FILES

/usr/games/lib/robots_roll the score file

BUGS

Bugs? You *crazy*, man?!?

NAME

rotcvph – rotate convex polyhedron

SYNOPSIS

/usr/demo/rotcvph*filename*

DESCRIPTION

rotcvph rotates a convex polyhedron with hidden surfaces removed. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. The mandatory file argument contains a polygonal object definition as described below.

Initially the program displays a fixed view of the object. The following commands may be typed at any time:

n Display successive views with no waiting.

w Wait for SPACE to be typed before displaying each view.

q Exit the program.

The format of the polygonal object definition is illustrated by this example of the definition of a pyramid:

```
        5       5
-1.0 1.0 -1.0 1.0 -1.0 1.0
1.0  1.0 -1.0
1.0 -1.0 -1.0
-1.0 -1.0 -1.0
-1.0  1.0 -1.0
0.0  0.0  1.0
4       4 3 2 1
3       1 5 4
3       2 5 1
3       3 5 2
3       4 5 3
```

The first line gives the number of vertices followed by the number of polygons. The second line gives the coordinates of a bounding box for the object. Minimum and maximum coordinate values are given for each of three dimensions in the order *minx*, *maxx*, *miny*, *maxy*, *minz*, *maxz*. Lines 3 through v+2 (where v is the number of vertices) give vertex coordinates in the order *x*, *y*, ,IR z . Lines v+3 through v+p+2 (where p is the number of polygons) give polygon descriptions. The first number is the number of vertices for the polygon. Succeeding numbers on the line are indices into the vertex list. Polygons should be planar. Coordinates are given in floating point format and everything else is integer. Entries on a given line are separated by arbitrary whitespace. A maximum of 400 vertices and 400 polygons may be defined. The polygon definitions may contain a maximum of 1600 instances of the vertices. **/usr/demo/data** contains several object definition files, including **icosa.dat**, **socbal.dat**, and **pyramid.dat**.

The above format may be used to define non-convex objects. The program will display these objects but hidden surface computations will not be done correctly.

FILES

/usr/demo/data/*.dat sample object definition files
icosa.dat
socbal.dat
pyramid.dat

BUGS

All floating point transformations are done twice for each view, once to draw the object and once to undraw it.

Lines which are common to two visible polygons in a view are drawn twice, once for each polygon.

NAME

snake, snscore – display chase game

SYNOPSIS

/usr/games/snake [–w*n*] [–l*n*]
/usr/games/snscore

DESCRIPTION

snake is a display-based game which must be played on a CRT terminal from among those supported by **vi**(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as **vi**(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc These keys are like hjkl but form a directed pad around the d key.

HJKL These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.

SEFC Likewise for the upper case versions on the left.

ATPB These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, for example, P is at the far right of the keyboard.

x This lets you quit the game at any time.

p Points in a direction you might want to go.

w Space warp to get out of tight squeezes, at a price.

! Shell escape

^Z Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run **/usr/games/snscore.**

FILES

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

soundtool – audio play/record tool

SYNOPSIS

soundtool

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

soundtool is a SunView demonstration program that allows recording, playing, and simple editing of audio data. The display consists of six regions: a play/record control panel, a function control panel, an oscilloscope, a display control panel, a waveform display panel, and a pop-up audio status panel.

Play/Record Control Panel

Play/Stop
: Clicking this button plays the currently selected region of data. While data is playing this button becomes a **Stop** button. If audio output is busy when **Play** is started, this button displays **Waiting**. When the device is available, the button switches to **Stop** and audio output begins. Clicking on the **Waiting** button resets the tool to the idle state.

Record/Stop
: Clicking this button starts the recording of data from the audio input port that is wired to the 8-pin mini-DIN connector on the back of SPARCstation 1 systems. While recording is in progress, this button becomes a **Stop** button. If audio input is busy when **Record** is selected, an alert pops up and the tool resets to the idle state. A maximum of 5 minutes may be recorded at a time.

Pause
: Clicking this button while playing or recording suspends the current operation. The button becomes a **Resume** button that may be selected to continue the suspended operation.

Describe
: Clicking this button brings up the "Audio Status Panel". If the panel was already visible, clicking this button removes it.

Quit
: Clicking this button exits **soundtool**.

Play Volume
: This slider adjusts the playback volume. Volume levels between 0 and 100 may be selected, where 0 represents infinite attenuation and 100 is maximum gain.

Record Volume
: This slider adjusts the recording level in the range 0 to 100.

Output To
: This selector switches the audio output port between the built-in speaker and the external headphone jack.

Looping
: When **Looping** is disabled, the current data region (that is, the data between the two markers in the waveform display) is played once. If **Looping** is enabled, the selected data plays endlessly until the **Stop** button is pressed.

Function Control Panel

Load
: Clicking **Load** reads in the audio file specified by the **Directory** and **File** fields. If the named file does not contain a valid audio header, the raw data is copied into the buffer and an alert is displayed. Clicking the **Store** button at that point rewrites the file with the proper audio file header.

: Arbitrarily large audio files may be loaded. However, system swap space resources may be depleted (one minute of SPARCstation 1 audio data consumes roughly .5 Mbyte of swap space).

Store Clicking **Store** writes the selected data region into the file specified by the **Directory** and **File** fields. If the named file exists, an alert will request confirmation of the operation.

Append
Clicking **Append** appends the selected data region to the file specified by the **Directory** and **File** fields. The named file must contain a valid audio file header.

Directory
The **Directory** field specifies a directory path in which to look for audio files.

File The **File** field designates the file to be loaded from, stored to, or appended to. Holding down the right mouse button on this field presents a menu of audio files in the currently designated directory. All files that contain a valid audio file header, or whose names have the suffix **.au** or **.snd**, are listed.

Oscilloscope
When the program is in the idle state and the cursor is in the waveform display panel, the oscilloscope acts as a magnifying glass, displaying the region of the audio waveform that is currently under the cursor. When the program is playing or recording, the oscilloscope displays the data that is currently being transferred. Note: there is a small time lag in the display of recorded data, due to the fact that the audio device driver buffers input data and delivers it to the application in discrete segments.

Display Control Panel

Zoom The **Zoom** slider adjusts the compression factor used in the display of the waveform. The upper compression limit is chosen so that the entire waveform fits in the waveform display panel. The lower limit is restricted by the ability to manipulate large scrolling regions in SunView. Adjustment of the **Zoom** slider ordinarily results in data compression or expansion around the center of the currently displayed waveform. If the waveform display contains one or both data selection markers, an attempt is made to keep at least a portion of the selected data region in the window.

The magnified waveform presented in the oscilloscope display is unaffected by the **Zoom** value. However, cursor movement over the waveform reflects the current compression; that is, lower **Zoom** values result in finer granularity of mouse movement.

Waveform Display Panel
The waveform display shows all or part of the current waveform, depending on the current **Zoom** value. Scrolling of the waveform may be achieved either by using the scrollbar or by dragging the waveform to the right or left while holding the middle mouse button down. Note: scrolling is disabled when the entire waveform is being displayed (that is, when the **Zoom** value is at its maximum).

In some cases, it is desirable to identify a subset of the waveform. For instance, the **Play**, **Store**, and **Append** functions operate on a selected region, rather than the entire waveform. The currently selected region of interest is delimited by dashed vertical lines. A new region may be selected by clicking the left or right mouse button and dragging it across the desired region of interest. Alternatively, a single click on the left or right mouse buttons adjusts the start or end points.

Audio Status Panel
This panel is displayed (or removed) when the **Describe** button is pressed. It contains fields that describe the data in the buffer.

Sample Rate
This field displays the sampling frequency, in samples per second.

Channels
This field denotes the number of interleaved channels of audio data.

Precision
This field identifies the encoding precision, in bits per sample.

Encoding

This field displays the encoding format.

Total Length

This field shows the length of the entire data buffer, in the form *hh:mm:ss.dd*.

Selection

This field identifies the start and end times of the currently selected region of interest.

Info String

When an audio file is loaded, the first 80 characters of the information field of the audio header are displayed in this field. This string may be edited, though the new information is only written out when the **Store** operation is performed.

BUGS

Currently, **soundtool** is capable of displaying only 8-bit μ-law encoded data. This restriction should be removed.

Audio files should be mapped in order to reduce the swap space requirements. The limit on recording length should also be removed.

SunView scrollbars operate on canvases whose virtual size is given by a short integer (that is, 16 bits). This ridiculous constraint is the reason for the lower limit on zooming. Because of this, the accuracy of start and end point selection is reduced when the data buffer is large.

Region selections made over the waveform display panel work best when the click and drag paradigm is used. Adjusting the start or end points by a single click is susceptible to error; that is, if the mouse moves slightly between the button down and up events, the result is a very small selection.

SEE ALSO

gaintool(6), **play**(6), **raw2audio**(6), **record**(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

suncoredemos – demonstrate SunCore Graphics Package

SYNOPSIS

/usr/demo/cproduct
/usr/demo/cshademo

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

suncoredemos is a collection of simple programs demonstrating the SunCore Graphics Package. Each program is briefly described below. These programs generate all graphics output using subroutine calls to SunCore. To exit each program, generate an interrupt signal by typing the appropriate key (usually DELETE).

cproduct Color Sun architecture demo (requires Sun Color Graphics Display).

cshademo Shaded surface polygons demo (requires Sun Color Graphics Display).

NAME

sunview_demos, canvas_demo, cursor_demo – Window-System demonstration programs

SYNOPSIS

/usr/demo/canvas_demo

/usr/demo/cursor_demo

AVAILABILITY

These demos are available with the *SunView Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

canvas_Demo

canvas_demo demonstrates the capabilities of the canvas subwindow package. It consists of two subwindows: a control panel and a canvas. By adjusting the items on the control panel, you can manipulate the attributes of the canvas, and see the results.

cursor_Demo

cursor_demo demonstrates what you can do with cursors. A single control panel is provide for adjusting the various cursor attributes. As you adjust the items on the control panel, the panel's cursor changes in appearance.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[**–a**] *filename*]

DESCRIPTION

trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commodore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

COMMAND SUMMARY

abandon	**ca**pture
cloak **u**p/**d**own	
computer request; ...	**da**mages
destruct	**do**ck
help	**i**mpulse course distance
lrscan	**m**ove course distance
phasers **a**utomatic amount	
phasers **m**anual amt1 course1 spread1 ...	
torpedo course [**y**es] angle/**n**o	
ram course distance	**r**est time
shell	**sh**ields **u**p/**d**own
srscan [**y**es/**n**o]	
status	**terminate** **y**es/**n**o
undock	**v**isual course
warp warp_factor	

NAME

vwcvph – view convex polyhedron

SYNOPSIS

/usr/demo/vwcvph *filename*

DESCRIPTION

vwcvph allows the user to view a convex polyhedron from various positions with hidden surfaces removed. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. The mandatory file argument contains a polygonal object definition as described in the manual page for **/usr/demo/rotcvph.**

The program operates in two modes: **DisplayObject** mode and **SelectView** mode. The program starts in **DisplayObject mode:**

DisplayObject:
: The object is displayed in 3-D perspective with hidden surfaces removed. Type **q** while in this mode to exit the program. Press RIGHT mouse button to switch to **SelectView** mode.

SelectView:
: Schematic projections of the outline of the object are shown and the mouse is used to select a viewing position. Press LEFT mouse button to set *x* and MIDDLE mouse button to set *y* in the *Front View*. Use MIDDLE mouse button to set *z* in the *Top View*. Press RIGHT mouse button to switch to **DisplayObject** mode.

The view shown in **DisplayObject** mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the object and that the positive *y* axis on the screen is the projection of the positive *y* axis in object coordinates.

The input file may define non-convex objects. The program will display these objects but hidden surface computations will not be done correctly.

FILES

/usr/demo/data/*.dat sample object definition files

BUGS

Lines which are common to two visible polygons in a view are drawn twice, once for each polygon.

NAME
worm – play the growing worm game

SYNOPSIS
/usr/games/worm [*size*]

DESCRIPTION
In **worm,** you are a little worm, your body is the o 's on the screen and your head is the @ . You move with the hjkl keys (as in the game **snake**). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit; if your worm eats the digit it will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS
If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms – animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**–field**] [**–length** #] [**–number** #] [**–trail**]

DESCRIPTION

–field makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

SEE ALSO

Snails by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt the wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2*, 2 (November 1973).

NAME

intro – miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for **troff**(1).

A 7V section number means one or more of the following:

- The man page documents System V behavior only.
- The man page documents default SunOS behavior, and System V behavior as it differs from the default behavior. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents behavior compliant with *IEEE Std 1003.1-1988* (POSIX.1).

LIST OF MISC. TABLES

Name	**Appears on Page**	**Description**
ansic	**ansic**(7V)	ANSI C (draft of December 7 1988) lint library
ascii	**ascii**(7)	map of ASCII character set
bsd	**bsd**(7)	overview of the Berkeley 4.3 environment
eqnchar	**eqnchar**(7)	special character definitions for eqn
filesystem	**filesystem**(7)	file system organization
hier	**hier**(7)	file system hierarchy
iso_8859_1	**iso_8859_1**(7)	map of character set
man	**man**(7)	macros to format Reference Manual pages
me	**me**(7)	macros for formatting papers
ms	**ms**(7)	text formatting macros
posix	**posix**(7V)	overview of the IEEE Std 1003.1-1988 (POSIX.1) environment
SunOS	**sunos**(7)	overview of the SunOS Release 4.1 environment
svidii	**svidii**(7V)	overview of the System V environment
svidiii	**svidiii**(7V)	SVIDIII lint library
xopen	**x/open**(7V)	overview of the XPG Issue 2 (X/Open) environment

NAME

ansic – ANSI C (draft of December 7 1988) lint library

SYNOPSIS

/usr/5bin/lint –n –lansic *ansic_src.c*

AVAILABILITY

This environment is not available under SunOS Release 4.1. The environment that most closely approximates an ANSI C environment is the System V environment. The System V environment is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

ANSI C is a proposed standard for the C language. SunOS Release 4.1 does not currently fully support ANSI C applications. It does support many of the functions described by the ANSI C draft. This man page does not imply that the functions supported by SunOS Release 4.1 and the functions described by the ANSI C draft perform identically. The ANSI C lint library is intended solely as a porting aid.

The ANSI C lint library consists exclusively of ANSI C functions. Users may lint their code with the **–n –lansic** options to catch all non-ANSI C features.

Certain functions defined in the ANSI C lint library are not available in the C library but are available. In particular, math functions are made available only when the **–lm** option is added to **cc**(1V) or **ld**(1) commands.

Other ANSI C functions not supported at all in SunOS Release 4.1 are **raise( )**, **fgetpos( )**, **fsetpos( )**, **div( )**, **ldiv( )**, **strtoul( )**, **strerror( )**, and **difftime( )**.

FILES

/usr/5lib/lint/llib-lansic*
ANSI C lint library

SEE ALSO

lint(1V), **bsd**(7), **posix**(7V), **sunos**(7), **svidii**(7V), **svidiii**(7V), **xopen**(7V)

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

/usr/pub/ascii is a map of the ASCII character set, to be printed as needed. It contains octal and hexadecimal values for each character. While not included in that file, a chart of decimal values is also shown here.

Octal — Character

000	NUL	001	SOH	002	STX	003	ETX	004	EOT	005	ENQ	006	ACK	007	BEL
010	BS	011	HT	012	NL	013	VT	014	NP	015	CR	016	SO	017	SI
020	DLE	021	DC1	022	DC2	023	DC3	024	DC4	025	NAK	026	SYN	027	ETB
030	CAN	031	EM	032	SUB	033	ESC	034	FS	035	GS	036	RS	037	US
040	SP	041	!	042	"	043	#	044	$	045	%	046	&	047	´
050	(	051	)	052	*	053	+	054	,	055	−	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	DEL

Hexadecimal — Character

00	NUL	01	SOH	02	STX	03	ETX	04	EOT	05	ENQ	06	ACK	07	BEL
08	BS	09	HT	0A	NL	0B	VT	0C	NP	0D	CR	0E	SO	0F	SI
10	DLE	11	DC1	12	DC2	13	DC3	14	DC4	15	NAK	16	SYN	17	ETB
18	CAN	19	EM	1A	SUB	1B	ESC	1C	FS	1D	GS	1E	RS	1F	US
20	SP	21	!	22	"	23	#	24	$	25	%	26	&	27	´
28	(	29	)	2A	*	2B	+	2C	,	2D	−	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~	7F	DEL

Decimal — Character

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	´
40	(	41	)	42	*	43	+	44	,	45	−	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL

FILES

/usr/pub/ascii Online chart of octal and hexadecimal values for the ASCII character set.

NAME

bsd – overview of the Berkeley 4.3 environment

SYNOPSIS

/usr/bin/lint −n −lbsd *bsd_src.c*

DESCRIPTION

BSD 4.3 is a set of functions and headers. The SunOS Release 4.1 is a superset of BSD 4.3. It includes all of the functionality described in the BSD 4.3 documentation. See **sunos**(7) for an overview of SunOS functionality.

Note: there may be some cases where the coexistence of another environment overrides the BSD 4.3 semantics. In particular, when there has been a point of conflict between POSIX.1 and BSD 4.3, POSIX.1 has won (see **setsid**(8V) for such an example).

Many man pages are marked with a "V" after the section number, indicating some sort of System V conformance. BSD 4.3 functions are also documented on these man pages, as well as on man pages without the "V" section suffix.

By default, the user will get a superset of the BSD 4.3 environment. No path modifications should be necessary. The typical path is **set path = (/usr/ucb /bin /usr/bin)**

LINT

As a portability aid, Sun is providing a lint library that consists exclusively of BSD 4.3 functions. Users may lint their code with the **−n −lbsd** options to catch all non-BSD 4.3 features.

BSD, as with most other environments, continues to evolve. The **−lbsd** lint library will always refer to the most recent BSD release supported by Sun. Some applications may wish to port to a particular release of BSD. They may safely use the more specific name of **−l4.3bsd** (currently the same as **−lbsd**). Lint libraries for BSD releases earlier than 4.3 are not currently available. 4.3 BSD is sufficiently close to 4.2 BSD that the 4.3 BSD lint library usually works.

FILES

/usr/bin/*	BSD 4.3 and SunOS specific executables
/usr/ucb/*	BSD 4.3 derived executables
/usr/include/*	BSD 4.3 and SunOS specific header files
/usr/lib/*	BSD 4.3 and SunOS specific library files
/usr/lib/lint/llib-lbsd*	BSD 4.3 lint library

SEE ALSO

lint(1V), **ansic**(7V), **posix**(7V), **sunos**(7), **svidii**(7V), **svidiii**(7V), **xopen**(7V), **setsid**(8V)

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [*filename*] | **troff** [*options*]

neqn /usr/pub/eqnchar [*filename*] | **nroff** [*options*]

DESCRIPTION

eqnchar contains **troff**(1) and **nroff**(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with **eqn**(1) and **eqn**(1). It contains definitions for the following characters

ciplus	⊕	*//*	//	*square*	□
citimes	⊗	*langle*	〈	*circle*	○
wig	~	*rangle*	〉	*blot*	□
-wig	≈	*hbar*	ħ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	*	*/<*	≮	*nomem*	∉
bigstar	✱	*/>*	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

eqn(1), **nroff**(1), **troff**(1)

NAME

filesystem – file system organization

SYNOPSIS

/

/usr

DESCRIPTION

The SunOS file system tree is organized for easy administration. Distinct areas within the file system tree are provided for files that are private to one machine, files that can be shared by multiple machines of a common architecture, files that can be shared by all machines, and home directories. This organization allows the sharable files to be stored on one machine, while being accessed by many machines using a remote file access mechanism such as Sun's Network File System (NFS). Grouping together similar files makes the file system tree easier to upgrade and manage.

The file system tree consists of a root file system and a collection of mountable file systems. The **mount**(8) program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system, or other previously mounted file systems. Two file systems, / (the root) and **/usr**, must be mounted in order to have a fully functional system. The root file system is mounted automatically by the kernel at boot time; the **/usr** file system is mounted by the **/etc/rc.boot** script, which is run as part of the booting process.

The root file system contains files that are unique to each machine; it can not be shared among machines. The root file system contains the following directories:

/dev Character and block special files. Device files provide hooks into hardware devices or operating system facilities. The **MAKEDEV** command (see **makedev**(8)) builds device files in the **/dev** directory. Typically, device files are built to match the kernel and hardware configuration of the machine.

/etc Various configuration files and system administration databases that are machine specific. You can think of **/etc** as the "home directory" of a machine, defining its "identity." Executable programs are no longer kept in **/etc**.

/home Mount points for home directories. This directory may be arranged so that shared user files are placed under the directory **/home**/*machine-name* on machines serving as file servers. Machines may then be locally configured with mount points under **/home** for all of the file servers of interest, with the name of the mount point being the name of the file server.

/mnt A generic mount point. This is an empty directory available for temporarily mounting file systems on.

/sbin Executable programs that are needed in the boot process before **/usr** is mounted. **/sbin** contains *only* those programs that are needed in order to mount the **/usr** file system: **hostname**(1), **ifconfig**(8C), **init**(8), **mount**(8), and **sh**(1). After **/usr** is mounted, the full complement of utilities are available.

/tmp Temporary files that are deleted at reboot time.

/var Files, such as log files, that are unique to a machine but that can grow to an arbitrary ("variable") size.

/var/adm System logging and accounting files.

/var/preserve
Backup files for **vi**(1) and **ex**(1).

/var/spool Subdirectories for files used in printer spooling, mail delivery, **cron**(8), **at**(1), etc.

/var/tmp Transitory files that are not deleted at reboot time.

Because it is desirable to keep the root file system small, larger file systems are often mounted on /**var** and /**tmp**.

The file system mounted on /**usr** contains architecture-dependent and architecture-independent shareable files. The subtree rooted at /**usr/share** contains architecture-independent shareable files; the rest of the /**usr** tree contains architecture-dependent files. By mounting a common remote file system, a group of machines with a common architecture may share a single /**usr** file system. A single /**usr/share** file system can be shared by machines of any architecture. A machine acting as a file server may export many different /**usr** file systems to support several different architectures and operating system releases. Clients usually mount /**usr** read-only to prevent their accidentally modifying any shared files. The /**usr** file system contains the following subdirectories:

/usr/5bin	System V executables.
/usr/5include	System V include files.
/usr/5lib	System V library files.
/usr/bin	Executable programs. The bulk of the system utilities are located here.
/usr/dict	Dictionary databases.
/usr/etc	Executable system administration programs.
/usr/games	Executable game programs and data.
/usr/include	Include files.
/usr/lib	Program libraries and various architecture-dependent databases.
/usr/pub	Various data files.
/usr/ucb	Executable programs descended from the Berkeley Software Distribution.
/usr/share	Subtree for architecture-independent shareable files.
/usr/share/man	Subdirectories for the on-line reference manual pages.
/usr/share/lib	Architecture-independent databases.

A machine with disks may export root file systems, swap files and /**usr** file systems to diskless or partially-disked machines, which mount these into the standard file system hierarchy. The standard directory tree for exporting these file systems is:

/export	The root of the exported file system tree.
/export/exec/*architecture-name*	The exported /**usr** file system supporting *architecture-name* for the current release.
/export/exec/*architecture-name.release-name*	The exported /**usr** file system supporting *architecture-name* for SunOS *release-name*.
/export/share	The exported common /**usr/share** directory tree.
/export/root/*hostname*	The exported root file system for *hostname*.
/export/swap/*hostname*	The exported swap file for *hostname*.
/export/var/*hostname*	The exported /**var** directory tree for *hostname*.
/export/dump/*hostname*	The exported dump file for *hostname*.
/export/crash/*hostname*	The exported crash dump directory for *hostname*.

Changes from Previous Releases

The file system layout described here is quite a bit different from the layout employed previous to release 4.0 of SunOS. For compatibility with earlier releases of SunOS, and other versions of the UNIX system, symbolic links are provided for various files and directories linking their previous names to their current locations. The symbolic links provided include:

/bin —> /usr/bin	All programs previously located in **/bin** are now in **/usr/bin**.
/lib —> /usr/lib	All files previously located in **/lib** are now in **/usr/lib**.
/usr/adm —> /var/adm	The entire **/usr/adm** directory has been moved to **/var/adm**.
/usr/spool —> /var/spool	The entire **/usr/spool** directory has been moved to **/var/spool**.
/usr/tmp —> /var/tmp	The **/usr/tmp** directory has been moved to **/var/tmp**.

/etc/termcap —> /usr/share/lib/termcap

/usr/5lib/terminfo —> /usr/share/lib/terminfo

/usr/lib/me —> /usr/share/lib/me

/usr/lib/ms —> /usr/share/lib/ms

/usr/lib/tmac —> /usr/share/lib/tmac

/usr/man —> /usr/share/man

The following program binaries have been moved from **/etc** to **/usr/etc** with symbolic links to them left in **/etc**: **arp**, **clri**, **cron**, **chown**, **chroot**, **config**, **dkinfo**, **dmesg**, **dump**, **fastboot**, **fasthalt**, **fsck**, **halt**, **ifconfig**, **link**, **mkfs**, **mknod**, **mount**, **ncheck**, **newfs**, **pstat**, **rdump**, **reboot**, **renice**, **restore**, **rmt**, **rrestore**, **shutdown**, **umount**, **update**, **unlink**, and **vipw**.

In addition, some files and directories have been moved with no symbolic link left behind in the old location:

Old Name	*New Name*
/etc/biod	**/usr/etc/biod**
/etc/fsirand	**/usr/etc/fsirand**
/etc/getty	**/usr/etc/getty**
/etc/in.rlogind	**/usr/etc/in.rlogind**
/etc/in.routed	**/usr/etc/in.routed**
/etc/in.rshd	**/usr/etc/in.rshd**
/etc/inetd	**/usr/etc/inetd**
/etc/init	**/usr/etc/init**
/etc/nfsd	**/usr/etc/nfsd**
/etc/portmap	**/usr/etc/portmap**
/etc/rpc.lockd	**/usr/etc/rpc.lockd**
/etc/rpc.statd	**/usr/etc/rpc.statd**
/etc/ypbind	**/usr/etc/ypbind**
/usr/lib/sendmail.cf	**/etc/sendmail.cf**
/usr/preserve	**/var/preserve**
/usr/lib/aliases	**/etc/aliases**
/stand	**/usr/stand**
/etc/yp	**/var/yp**

Note: with this new file system organization, the approach to repairing a broken file system changes. One must mount **/usr** before doing an **fsck**(8), for example. If the mount point for **/usr** has been destroyed, **/usr** can be mounted temporarily on **/mnt** or **/tmp**. If the root file system on a standalone system is so badly damaged that none of these mount points exist, or if **/sbin/mount** has been corrupted, the only way to repair it may be to re-install the root file system.

SEE ALSO

at(1), **ex**(1), **hostname**(1), **sh**(1), **vi**(1), **intro**(4), **nfs**(4P), **hier**(7), **fsck**(8), **ifconfig**(8C), **init**(8), **makedev**(8), **mount**(8), **rc**(8)

NAME

hier – file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a typical SunOS file system hierarchy:

/ root directory of the file system

/dev/ devices (Section 4)

- **MAKEDEV** shell script to create special files
- **MAKEDEV.local** site specific part of **MAKEDEV**
- **console** main system console, **console**(4S)
- **drum** paging device, **drum**(4)
- ***mem** memory special files, **mem**(4S)
- **null** null file or data sink, **null**(4)
- **pty[p-z]*** pseudo terminal controllers, **pty**(4)
- **tty[ab]** CPU serial ports, **zs**(4S)
- **tty[0123][0-f]** MTI serial ports **mti**(4S)
- **tty[hijk][0-f]** ALM-2 serial ports **mcp**(4S)
- **tty[p-z]*** pseudo terminals, **pty**(4)
- **vme*** VME bus special files, **mem**(4S)
- **win** window system special files, **win**(4S)
- **xy*** disks, **xy**(4S)
- **rxy*** raw disk interfaces, **xy**(4S)
- . . .

/etc/ system-specific maintenance and data files

- **dumpdates** dump history, **dump**(8)
- **exports** table of file systems exportable with NFS, **exports**(5)
- **fstab** file system configuration table, **fstab**(5)
- **group** group file, **group**(5)
- **hosts** host name to network address mapping file, **hosts**(5)
- **hosts.equiv** list of trusted systems, **hosts.equiv**(5)
- **motd** message of the day, **login**(1)
- **mtab** mounted file table, **mtab**(5)
- **networks** network name to network number mapping file, **networks**(5)
- **passwd** password file, **passwd**(5)
- **phones** private phone numbers for remote hosts, as described in **phones**(5)
- **printcap** table of printers and capabilities, **printcap**(5)
- **protocols** protocol name to protocol number mapping file, **protocols**(5)
- **rc** shell program to bring the system up multiuser
- **rc.boot** startup file run at boot time
- **rc.local** site dependent portion of **rc**
- **remote** names and description of remote hosts for **tip**(1C), **remote**(5)
- **services** network services definition file, **services**(5)

ttytab database of terminal information used by **getty**(8)

...

/export/

directory of exported files and file systems for clients, including swap files, root, and **/usr** file systems

/home/ directory of mount points for remote-mounted home directories and shared file systems

user home (initial working) directory for *user*

.profile set environment for **sh**(1), **environ**(5V)

.project

what you are doing (used by (**finger**(1))

.cshrc startup file for **csh**(1)

.exrc startup file for **ex**(1)

.plan what your short-term plans are (used by **finger**(1))

.rhosts host equivalence file for **rlogin**(1C)

.mailrc startup file for **mail**(1)

calendar

user's datebook for **calendar**(1)

...

/lost+found

directory for connecting detached files for **fsck**(8)

/mnt/ mount point for file systems mounted temporarily

/sbin/ executable programs needed to mount **/usr/**

hostname

ifconfig

init

mount

sh

/tmp/ temporary files, usually on a fast device, see also /**var/tmp/**

ctm* used by **cc**(1V)

e* used by **ed**(1)

...

/var/ directory of files that tend to grow or vary in size

adm/ administrative log files

lastlog record of recent logins, **utmp**(5V)

lpacct line printer accounting **lpr**(1)

messages

system messages

tracct phototypesetter accounting, **troff**(1)

utmp table of currently logged in users, **utmp**(5V)

vaacct, vpacct

varian and versatec accounting **vtroff**(1), **pac**(8)

wtmp login history, **utmp**(5V)

...

preserve/

editor temporaries preserved here after crashes/hangups

spool/ delayed execution files

cron/ used by **cron**(8)

lpd/ used by **lpr**(1)

lock present when line printer is active

cf* copy of file to be printed, if necessary

df* control file for print job

tf* transient control file, while *lpr* is working

mail/ mailboxes for **mail**(1)
name mail file for user *name*
name.**lock**
lock file while *name* is receiving mail
mqueue/
mail queue for **sendmail**(8)
secretmail/
like **mail/**, but used by **xsend**(1)
uucp/ work files and staging area for **uucp**(1C)
LOGFILE
summary log
LOG.* log file for one transaction
...
tmp/ temporary files, to keep **/tmp/** small
raster used by **plot**(1G)
stm* used by **sort**(1V)
...
yp/ Network Information Service (NIS) database files, **ypfiles**(5)
/usr/ general-purpose directory, usually a mounted file system
bin/ utility programs
as assembler, **as**(1)
cc C compiler executive, c.f. **/usr/lib/ccom**, **/usr/lib/cpp**, **/usr/lib/c2**
csh the C-shell, **csh**(1)
sh the Bourne shell, **sh**(1)
...
demo/ demonstration programs
diag/ system tests and diagnostics
dict/ word lists, etc.
spellhist
history file for **spell**(1)
words principal word list, used by **look**(1)
...
etc/ system administration programs; c.f. section 8
catman update preformatted man pages, **catman**(8)
cron the clock daemon, **cron**(8)
dump file system backup program **dump**(8)
getty part of **login**(1), **getty**(8)
in.comsat
biff server (incoming mail daemon), **comsat**(8C)
init the parent of all processes, **init**(8)
mount **mount**(8)
yp/ NIS programs
ypinit build and install NIS database, **ypinit**(8)
yppush force propagation of a changed NIS map, **yppush**(8)
ypset point **ypbind** at a particular server, **ypset**(8)
...
...
games/
backgammon

- **lib/** library directory for game scores, etc.
  - **quiz.k/** what **quiz**(6) knows
    - **africa** countries and capitals
    - **index** category index
    - ...
  - ...
- ...

hosts/ symbolic links to **rsh**(1C) for commonly accessed remote hosts

include/

- standard **#include** files
- **a.out.h** object file layout, **a.out**(5)
- **images/** icon images
- **machine/**
  - header files from **/usr/share/sys/sys/machine**; may be a symbolic link
- **math.h** **intro**(3M)
- **net/** header files from **/usr/share/sys/sys/net**; may be a symbolic link
- **nfs/** header files used in the Network File System (NFS)
- **stdio.h** standard I/O, **intro**(3)
- **sys/** kernel header files, c.f. **/usr/share/sys/sys**
- ...

lib/ object libraries, compiler program binaries, and other data

- **ccom** C compiler proper
- **cpp** C preprocessor
- **c2** C code improver
- **eign** list of English words to be ignored by **ptx**(1)
- **font/** fonts for **troff**(1)
  - **ftR** Times Roman
  - **ftB** Times Bold
  - ...
- **libc.a** system calls, standard I/O, etc. (2,3,3S)
- **libm.a** math library, **intro**(3M)
- **lint/** utility files for lint
  - **lint[12]** subprocesses for **lint**(1V)
  - **llib-lc** dummy declarations for **/usr/lib/libc.a**, used by **lint**(1V)
  - **llib-lm** dummy declarations for **/usr/lib/libm.a**
  - ...
- **units** conversion tables for **units**(1)
- **uucp/** programs and data for **uucp**(1C)
  - **L.sys** remote system names and numbers
  - **uucico** the real copy program
  - ...
- ...

local/ locally maintained software

old/ obsolete and unsupported programs

pub/ publicly readable data files

sccs/ binaries of programs that compose the source code control system (SCCS)

src/ system source code tree

stand/ standalone programs (not run under the Sun Operating System)

share/ architecture independent files

- **lib/** architecture independent data files
  - **termcap**
    - description of terminal capabilities, **termcap**(5)

tmac/ macros for **troff**(1)
tmac.an
macros for **man**(7)
tmac.s macros for **ms**(7)
. . .
. . .
man/ on-line reference manual pages, **man**(1)
man?/ source files (**nroff**(1)) for sections 1 through 8 of the manual
as.1
. . .
cat?/ preformatted pages for sections 1 through 8 of the manual
. . .
sys/ SunOS kernel source and object modules
ucb/ binaries of programs developed at the University of California, Berkeley
ex line-oriented editor for experienced users, **ex**(1)
vi screen-oriented editor, **vi**(1)
. . .

/vmunix
the SunOS kernel binary

SEE ALSO

filesystem(7), **find**(1), **finger**(1), **grep**(1V), **ls**(1V), **rlogin**(1C), **whatis**(1), **whereis**(1), **which**(1), **ncheck**(8)

BUGS

The locations of files are subject to change without notice; the organization of your file system may vary.

This list is incomplete.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

iso_8859_1 – map of character set

SYNOPSIS

cat /usr/share/lib/locale/LC_CTYPE/iso_8859_1

DESCRIPTION

/usr/share/lib/locale/LC_CTYPE/iso_8859_1 is a map of the ISO_8859/1 character set, to be printed as needed.

This character set is available if **setlocale (3V)** is declared as:

> **setlocale(LC_CTYPE, iso_8859_1)**

or:

> **setlocale(LC_ALL, iso_8859_1)** see **setlocale**(3V) for more information about declaring categories and locales.

ISO Latin 1 Character Set

The following table displays the ISO 8859/1 character set.

ISO Latin 1				
Row/Col	Decimal	Octal		Name
02/00	032	040	SP	SPACE
02/01	033	041	!	EXCLAMATION POINT
02/02	034	042	"	QUOTATION MARK
02/03	035	043	#	NUMBER SIGN
02/04	036	044	$	DOLLAR SIGN
02/05	037	045	%	PERCENT SIGN
02/06	038	046	&	AMPERSAND
02/07	039	047	'	APOSTROPHE
02/08	040	050	(	LEFT PARENTHESIS
02/09	041	051	)	RIGHT PARENTHESIS
02/10	042	052	*	ASTERISK
02/11	043	053	+	PLUS SIGN
02/12	044	054	,	COMMA
02/13	045	055	-	HYPHEN, MINUS SIGN
02/14	046	056	.	FULL STOP (U.S.: PERIOD, DECIMAL POINT)
02/15	047	057	/	SOLIDUS (U.S.: SLASH)
03/00	048	060	0	DIGIT ZERO
03/01	049	061	1	DIGIT ONE
03/02	050	062	2	DIGIT TWO
03/03	051	063	3	DIGIT THREE
03/04	052	064	4	DIGIT FOUR
03/05	053	065	5	DIGIT FIVE
03/06	054	066	6	DIGIT SIX
03/07	055	067	7	DIGIT SEVEN
03/08	056	070	8	DIGIT EIGHT
03/09	057	071	9	DIGIT NINE
03/10	058	072	:	COLON
03/11	059	073		
03/12	060	074	<	LESS-THAN SIGN
03/13	061	075	=	EQUALS SIGN
03/14	062	076	>	GREATER-THAN SIGN
03/15	063	077	?	QUESTION MARK

ISO Latin 1 (continued)				
Row/Col	Decimal	Octal		Name
04/00	064	100	@	COMMERCIAL AT
04/01	065	101	A	LATIN CAPITAL LETTER A
04/02	066	102	B	LATIN CAPITAL LETTER B
04/03	067	103	C	LATIN CAPITAL LETTER C
04/04	068	104	D	LATIN CAPITAL LETTER D
04/05	069	105	E	LATIN CAPITAL LETTER E
04/06	070	106	F	LATIN CAPITAL LETTER F
04/07	071	107	G	LATIN CAPITAL LETTER G
04/08	072	110	H	LATIN CAPITAL LETTER H
04/09	073	111	I	LATIN CAPITAL LETTER I
04/10	074	112	J	LATIN CAPITAL LETTER J
04/11	075	113	K	LATIN CAPITAL LETTER K
04/12	076	114	L	LATIN CAPITAL LETTER L
04/13	077	115	M	LATIN CAPITAL LETTER M
04/14	078	116	N	LATIN CAPITAL LETTER N
04/15	079	117	O	LATIN CAPITAL LETTER O
05/00	080	120	P	LATIN CAPITAL LETTER P
05/01	081	121	Q	LATIN CAPITAL LETTER Q
05/02	082	122	R	LATIN CAPITAL LETTER R
05/03	083	123	S	LATIN CAPITAL LETTER S
05/04	084	124	T	LATIN CAPITAL LETTER T
05/05	085	125	U	LATIN CAPITAL LETTER U
05/06	086	126	V	LATIN CAPITAL LETTER V
05/07	087	127	W	LATIN CAPITAL LETTER W
05/08	088	130	X	LATIN CAPITAL LETTER X
05/09	089	131	Y	LATIN CAPITAL LETTER Y
05/10	090	132	Z	LATIN CAPITAL LETTER Z
05/11	091	133	[	LEFT SQUARE BRACKET
05/12	092	134	\	REVERSE SOLIDUS (U.S.: BACK SLASH)
05/13	093	135	]	RIGHT SQUARE BRACKET
05/14	094	136	^	CIRCUMFLEX ACCENT
05/15	095	137	_	LOW LINE (U.S.: UNDERSCORE)
06/00	096	140	`	GRAVE ACCENT
06/01	097	141	a	LATIN SMALL LETTER a
06/02	098	142	b	LATIN SMALL LETTER b
06/03	099	143	c	LATIN SMALL LETTER c
06/04	100	144	d	LATIN SMALL LETTER d
06/05	101	145	e	LATIN SMALL LETTER e
06/06	102	146	f	LATIN SMALL LETTER f
06/07	103	147	g	LATIN SMALL LETTER g
06/08	104	150	h	LATIN SMALL LETTER h
06/09	105	151	i	LATIN SMALL LETTER i
06/10	106	152	j	LATIN SMALL LETTER j
06/11	107	153	k	LATIN SMALL LETTER k
06/12	108	154	l	LATIN SMALL LETTER l
06/13	109	155	m	LATIN SMALL LETTER m
06/14	110	156	n	LATIN SMALL LETTER n
06/15	111	157	o	LATIN SMALL LETTER o

ISO Latin 1 (continued)				
Row/Col	Decimal	Octal		Name
07/00	112	160	p	LATIN SMALL LETTER p
07/01	113	161	q	LATIN SMALL LETTER q
07/02	114	162	r	LATIN SMALL LETTER r
07/03	115	163	s	LATIN SMALL LETTER s
07/04	116	164	t	LATIN SMALL LETTER t
07/05	117	165	u	LATIN SMALL LETTER u
07/06	118	166	v	LATIN SMALL LETTER v
07/07	119	167	w	LATIN SMALL LETTER w
07/08	120	170	x	LATIN SMALL LETTER x
07/09	121	171	y	LATIN SMALL LETTER y
07/10	122	172	z	LATIN SMALL LETTER z
07/11	123	173	{	LEFT CURLY BRACKET
07/12	124	174	|	VERTICAL LINE
07/13	125	175	}	RIGHT CURLY BRACKET
07/14	126	176	~	TILDE
10/00	160	240		NO-BREAK SPACE
10/01	161	241		INVERTED EXCLAMATION MARK
10/02	162	242		CENT SIGN
10/03	163	243		POUND SIGN
10/04	164	244		CURRENCY SIGN
10/05	165	245		YEN SIGN
10/06	166	246		BROKEN BAR
10/07	167	247		PARAGRAPH SIGN, (U.S.: SECTION SIGN)
10/08	168	250		DIAERESIS
10/09	169	251		COPYRIGHT SIGN
10/10	170	252		FEMININE ORDINAL INDICATOR
10/11	171	253		LEFT ANGLE QUOTATION MARK
10/12	172	254		NOT SIGN
10/13	173	255		SHY SOFT HYPHEN
10/14	174	256		REGISTERED TRADEMARK SIGN
10/15	175	257		MACRON
11/00	176	260		RING ABOVE, DEGREE SIGN
11/01	177	261		PLUS-MINUS SIGN
11/02	178	262		SUPERSCRIPT TWO
11/03	179	263		SUPERSCRIPT THREE
11/04	180	264		ACUTE ACCENT
11/05	181	265		MICRO SIGN
11/06	182	266		PILCROW SIGN, (U.S.: PARAGRAPH)
11/07	183	267		MIDDLE DOT
11/08	184	270		CEDILLA
11/09	185	271		SUPERSCRIPT ONE
11/10	186	272		MASCULINE ORDINAL INDICATOR
11/11	187	273		RIGHT ANGLE QUOTATION MARK
11/12	188	274		VULGAR FRACTION ONE QUARTER
11/13	189	275		VULGAR FRACTION ONE HALF
11/14	190	276		VULGAR FRACTION THREE QUARTERS
11/15	191	277		INVERTED QUESTION MARK

ISO Latin 1 (continued)

Row/Col	Decimal	Octal		Name
12/00	192	300		LATIN CAPITAL LETTER A WITH GRAVE ACCENT
12/01	193	301		LATIN CAPITAL LETTER A WITH ACUTE ACCENT
12/02	194	302		LATIN CAPITAL LETTER A WITH CIRCUMFLEX ACCENT
12/03	195	303		LATIN CAPITAL LETTER A WITH TILDE
12/04	196	304		LATIN CAPITAL LETTER A WITH DIAERESIS
12/05	197	305		LATIN CAPITAL LETTER A WITH RING ABOVE
12/06	198	306		CAPITAL DIPHTHONG AE
12/07	199	307		LATIN CAPITAL LETTER C WITH CEDILLA
12/08	200	310		LATIN CAPITAL LETTER E WITH GRAVE ACCENT
12/09	201	311		LATIN CAPITAL LETTER E WITH ACUTE ACCENT
12/10	202	312		LATIN CAPITAL LETTER E WITH CIRCUMFLEX ACCENT
12/11	203	313		LATIN CAPITAL LETTER E WITH DIAERESIS
12/12	204	314		LATIN CAPITAL LETTER I WITH GRAVE ACCENT
12/13	205	315		LATIN CAPITAL LETTER I WITH ACUTE ACCENT
12/14	206	316		LATIN CAPITAL LETTER I WITH CIRCUMFLEX ACCENT
12/15	207	317		LATIN CAPITAL LETTER I WITH DIAERESIS
13/00	208	320		CAPITAL ICELANDIC LETTER ETH
13/01	209	321		LATIN CAPITAL LETTER N WITH TILDE
13/02	210	322		LATIN CAPITAL LETTER O WITH GRAVE ACCENT
13/03	211	323		LATIN CAPITAL LETTER O WITH ACUTE ACCENT
13/04	212	324		LATIN CAPITAL LETTER O WITH CIRCUMFLEX ACCENT
13/05	213	325		LATIN CAPITAL LETTER O WITH TILDE
13/06	214	326		LATIN CAPITAL LETTER O WITH DIAERESIS
13/07	215	327		MULTIPLICATION SIGN
13/08	216	330		LATIN CAPITAL LETTER O WITH OBLIQUE STROKE
13/09	217	331		LATIN CAPITAL LETTER U WITH GRAVE ACCENT
13/10	218	332		LATIN CAPITAL LETTER U WITH ACUTE ACCENT
13/11	219	333		LATIN CAPITAL LETTER U WITH CIRCUMFLEX
13/12	220	334		LATIN CAPITAL LETTER U WITH DIAERESIS
13/13	221	335		LATIN CAPITAL LETTER Y WITH ACUTE ACCENT
13/14	222	336		CAPITAL ICELANDIC LETTER THORN
13/15	223	337		SMALL GERMAN LETTER SHARP s
14/00	224	340		LATIN SMALL LETTER a WITH GRAVE ACCENT
14/01	225	341		LATIN SMALL LETTER a WITH ACUTE ACCENT
14/02	226	342		LATIN SMALL LETTER a WITH CIRCUMFLEX ACCENT
14/03	227	343		LATIN SMALL LETTER a WITH TILDE
14/04	228	344		LATIN SMALL LETTER a WITH DIAERESIS
14/05	229	345		LATIN SMALL LETTER a WITH RING ABOVE
14/06	230	346		SMALL DIPHTHONG ae
14/07	231	347		LATIN SMALL LETTER c WITH CEDILLA
14/08	232	350		LATIN SMALL LETTER e WITH GRAVE ACCENT
14/09	233	351		LATIN SMALL LETTER e WITH ACUTE ACCENT
14/10	234	352		LATIN SMALL LETTER e WITH CIRCUMFLEX ACCENT
14/11	235	353		LATIN SMALL LETTER e WITH DIAERESIS
14/12	236	354		LATIN SMALL LETTER i WITH GRAVE ACCENT
14/13	237	355		LATIN SMALL LETTER i WITH ACUTE ACCENT
14/14	238	356		LATIN SMALL LETTER i WITH CIRCUMFLEX ACCENT
14/15	239	357		LATIN SMALL LETTER i WITH DIAERESIS

ISO Latin 1 (continued)				
Row/Col	Decimal	Octal		Name
15/00	240	360		SMALL ICELANDIC LETTER ETH
15/01	241	361		LATIN SMALL LETTER n WITH TILDE
15/02	242	362		LATIN SMALL LETTER o WITH GRAVE ACCENT
15/03	243	363		LATIN SMALL LETTER o WITH ACUTE ACCENT
15/04	244	364		LATIN SMALL LETTER o WITH CIRCUMFLEX ACCENT
15/05	245	365		LATIN SMALL LETTER o WITH TILDE
15/06	246	366		LATIN SMALL LETTER o WITH DIAERESIS
15/07	247	367		DIVISION SIGN
15/08	248	370		LATIN SMALL LETTER o WITH OBLIQUE STROKE
15/09	249	371		LATIN SMALL LETTER u WITH GRAVE ACCENT
15/10	250	372		LATIN SMALL LETTER u WITH ACUTE ACCENT
15/11	251	373		LATIN SMALL LETTER u WITH CIRCUMFLEX ACCENT
15/12	252	374		LATIN SMALL LETTER u WITH DIAERESIS
15/13	253	375		LATIN SMALL LETTER y WITH ACUTE ACCENT
15/14	254	376		SMALL ICELANDIC LETTER THORN
15/15	255	377		LATIN SMALL LETTER y WITH DIAERESIS

SEE ALSO

setlocale(3V)

NAME

man – macros to format Reference Manual pages

SYNOPSIS

nroff −man *filename*...

troff −man *filename*...

DESCRIPTION

These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by **man**(1). See **Conventions**.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way **.I** may be used to italicize a whole line, or **.SB** may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by −**man**:

*R	'®', '(Reg)' in **nroff.**
*S	Change to default type size.

Requests

Request	*Cause Break*	*If no Argument*	*Explanation*
.B *t*	no	*t*=n.t.l.*	Text is in bold font.
.BI *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words, alternating bold and roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
.I *t*	no	*t*=n.t.l.	Text is italic.
.IB *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as **.TP** with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words, alternating italic and roman.
.IX *t*	no	-	Index macro, for Sun internal use.
.LP	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
.PD *d*	no	*d*=.4v	Set vertical distance between paragraphs.
.PP	yes	-	Same as **.LP**.
.RE	yes	-	End of relative indent. Restores prevailing indent.
.RB *t*	no	*t*=n.t.l.	Join words, alternating roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words, alternating roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
.SB *t*	no	-	Reduce size of text by 1 point, make text boldface.
.SH *t*	yes	-	Section Heading.
.SM *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
.SS *t*	yes	*t*=n.t.l.	Section Subheading.

.TH *n s d f m*	yes	-	Begin reference page *n*, of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
.TX *t p*	no	-	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*. *

n.t.l. = next text line; p.i. = prevailing indent

Conventions

When formatting a manual page, **man** examines the first line to determine whether it requires special processing. For example a first line consisting of:

'\" t

indicates that the manual page must be run through the **tbl**(1) preprocessor.

A typical manual page for a SunOS command or function is laid out as follows:

.TH *TITLE* [1-8]
The name of the command or function in upper-case, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no **troff**(1) commands or escapes, and no macro requests. It is used to generate the **whatis**(1) database.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.

... Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION
A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in boldface, as do literal filenames and references to items that appear elsewhere in the *SunOS Reference Manual.* Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section only describes the behavior of the command itself, not that of subcommands.

.SH OPTIONS
The list of options along with a description of how each affects the command's operation.

.SH FILES
A list of files associated with the command or function.

.SH SEE ALSO
A comma-separated list of related manual pages, followed by references to other published materials.

.SH DIAGNOSTICS
A list of diagnostic messages and an explanation of each.

.SH BUGS
A description of limitations, known defects, and possible problems associated with the command or function.

FILES

/usr/share/lib/tmac/tmac.an

SEE ALSO

man(1), **nroff**(1), **troff**(1), **whatis**(1)

Formatting Documents.

NAME

me – macros for formatting papers

SYNOPSIS

nroff –me [options] file ...
troff –me [options] file ...

DESCRIPTION

This package of **nroff** and **troff** macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many **nroff** and **troff** requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the **eqn, neqn, refer,** and **tbl**(1) preprocessors for equations and tables is acceptable as input.

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x*x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m defines the part of the paper, and can be* **C** (chapter), **A** (appendix), **P** (preliminary, for instance, abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by **eqn** or **meqn**.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.

.PE	-	yes	End *pic* picture.
.PS	-	yes	Begin *pic* picture.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bu	-	yes	Begin bulleted paragraph
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *´x´y´z*	'''''	no	Set even footer to x y z
.eh *´x´y´z*	'''''	no	Set even header to x y z
.fo *´x´y´z*	'''''	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *´x´y´z*	'''''	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form .*x. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *´x´y´z*	'''''	no	Set odd footer to x y z
.oh *´x´y´z*	'''''	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sm *x* -	*no*	*Set x* in a smaller pointsize.	
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in **troff**). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

FILES

/usr/share/lib/tmac/tmac.e
/usr/share/lib/me/*

SEE ALSO

eqn(1), **nroff**(1), **troff**(1), **refer**(1), **tbl**(1)

Formatting Documents

NAME

ms – text formatting macros

SYNOPSIS

nroff –ms [*options*] *filename* ...

troff –ms [*options*] *filename* ...

DESCRIPTION

This package of **nroff**(1) and **troff**(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through **col**(1V). All external **–ms** macros are defined below.

Note: this **–ms** macro package is an extended version written at Berkeley and is a superset of the standard **–ms** macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many **nroff** and **troff** requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp *n*	insert n spacing lines
.ce *n*	center next n lines
.ls *n*	line spacing: *n*=**1** single, *n*=**2** double space
.na	no alignment of right margin

Font and point size changes with **\f** and **\s** are also allowed; for example, **\fIword\fR** will italicize *word*. Output of the **tbl**(1), **eqn**(1) and **refer**(1) preprocessors for equations, tables, and references is acceptable as input.

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no do not label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I, L, C, B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for **.tl**)
.EH *x*	–	n	even page header *x* (3 part as for **.tl**)
.EN	–	y	end displayed equation produced by **eqn**

.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for **.tl**)
.OH *x*	–	n	odd page header *x* (3 part as for **.tl**)
.P1	if TM	n	print header on first page
.PP	–	y,y	paragraph with first line indented
.PT	- -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on first page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set TAB characters to 8n 16n ... (**nroff**) 5n 10n ... (**troff**)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by **tbl**
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in **troff**
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of **.IX** entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]–	–	n	beginning of **refer** reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in **–ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

.nr LL 6.5i

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as **LL**
FL	footnote length	next **.FS**	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next **.QP**	5n
FI	footnote indent	next **.FS**	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next **.FS**	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting **FF** to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an **.IP**-like footnote paragraph.

Here is a list of string registers available in **–ms**; they may be used anywhere in the text:

Name	String's Function
***Q**	quote (" in **nroff,** `` in **troff**)
***U**	unquote (" in **nroff,** '' in **troff**)
***–**	dash (-- in **nroff,** — in **troff**)
***(MO**	month (month of the year)
***(DY**	day (current date)
******	automatically numbered footnote
***´**	acute accent (before letter)
***`**	grave accent (before letter)
***^**	circumflex (before letter)
***,**	cedilla (before letter)
***:**	umlaut (before letter)
***~**	tilde (before letter)

When using the extended accent mark definitions available with **.AM**, these strings should come after, rather than before, the letter to be accented.

FILES

/usr/share/lib/tmac/tmac.s
/usr/share/lib/ms/ms.???

SEE ALSO

col(1V), **eqn**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1)

Formatting Documents

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

posix – overview of the IEEE Std 1003.1-1988 (POSIX.1) environment

SYNOPSIS

/usr/5bin/lint –n –lposix *posix_src.c*

AVAILABILITY

This environment is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

POSIX.1 is a set of functions and headers. The SunOS Release 4.1 implementation of POSIX.1 is a superset — it includes all of the functionality described in the IEEE standard as well as most of the SunOS functionality. See the **sunos(7)** man page for a description of SunOS functionality.

All man pages that are associated with POSIX.1 are marked by a "V" after the section number. Not all "V" pages, however, are POSIX.1. Some "V" pages may be part of other System V based environments such as X/Open.

If a user desires to work in a POSIX.1 (or System V) environment, the user should set the path variable to include **/usr/5bin** before anything else. The typical path is **PATH=/usr/5bin:/bin:/usr/bin:/usr/ucb**.

LINT

As a portability aid, Sun is providing a lint library that consists exclusively of POSIX.1 functions. Users may lint their code with the **–n –lposix** options to catch all non-POSIX.1 features.

POSIX.1 is primarily an operating system interface. POSIX.1 also specifies a subset of the functions defined by ANSI C. These are included in the **posix** lint library. Because of the additional functionality provided by ANSI C, Sun will also be providing an ANSI C (based on the December 7, 1988 draft) lint library. A portable application may want to lint with **–n –lposix –lansic** for the most complete coverage of functions.

POSIX.1 as with most other environments, continues to evolve. The **–lposix** lint library will always refer to the most recent standard supported by Sun. Some applications may wish to port to a particular version of the standard; they may safely use the more specific name of **–lposix1-88** (currently the same as **–lposix**).

Certain functions defined in the **posix** lint library are not available in the C library. In particular, math functions are made available only when the **–lm** option is added to **cc**(1V) or **ld**(1) commands.

FILES

/usr/5bin/*	POSIX.1 and System V specific executables
/usr/5include/*	POSIX.1 and System V specific headers
/usr/5lib/*	POSIX.1 and System V specific library files

SEE ALSO

lint(1V), **ansic**(7V), **bsd**(7), **sunos**(7), **svidii**(7V), **svidiii**(7V), **xopen**(7V)

IEEE Std 1003.1-1988

NAME

sunos, SunOS – overview of the SunOS Release 4.1 environment

SYNOPSIS

lint *sunos_src.c*

DESCRIPTION

The SunOS Release 4.1 lint library is a superset of the 4.3 BSD lint library. It includes all of the 4.3 BSD functionality, most of System V release 3.2 functionality, as well as extensive additional functionality in the networking and file system areas.

It is important to note that the default environment in SunOS Release 4.1 provides BSD 4.3 compatibility. Sun also provides a System V compatible environment (see **svidii**(7V)).

Note that many man pages are marked with a "V" after the section number, indicating some sort of System V compliance. SunOS functions are also documented on these man pages, as well as on man pages without the "V" section suffix.

By default, the user will get the SunOS environment. No path modifications should be necessary. The typical path is **set path** = (**/bin /usr/bin /usr/ucb**)

FILES

/usr/bin/*	SunOS executables
/usr/ucb/*	BSD derived executables
/usr/include/*	SunOS specific header files
/usr/lib/*	SunOS specific library files

SEE ALSO

lint(1V), **ansic**(7V), **bsd**(7), **posix**(7V), **svidii**(7V), **svidiii**(7V), **xopen**(7V)

NAME

svidii – overview of the System V environment

SYNOPSIS

/usr/5bin/lint –n –lsvidii *sys5_src.c*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

SVID II is a set of functions and header files. The SunOS Release 4.1 implementation of SVID II is a superset — it includes all of the functionality described in the SVID issue 2 documents as well as most of the SunOS functionality. See the **sunos**(7) man page for a description of SunOS functionality.

All man pages that are associated with SVID II are marked by a "V" after the section number. Not all "V" pages are SVID II, however. Some "V" pages may be part of other System V based environments such as X/Open.

If a user desires to work in a SVID II environment, the user should set the path variable to include **/usr/xpg2bin** and **/usr/5bin** before anything else. The typical path is:

set path=(/usr/xpg2bin /usr/5bin /bin /usr/bin /usr/ucb)

As a portability aid, Sun is providing two lint libraries that consist exclusively of SVID II functions as defined in the SVID issue 2. Users may lint their code with the **–n –lsvidii** options to catch all features that are not found in SVID issue 2, all volumes. Using lint with the **–n –lsvidii-3** options is just like **–n –lsvidii** except that it does not include volume 3 (which contains new directory reading routines and new signal functions that appeared in System V release 3.2).

FILES

/usr/5bin/*	System V specific executables
/usr/5include/*	System V specific header files
/usr/5lib/*	System V specific library files

SEE ALSO

lint(1V), **ansic**(7V), **bsd**(7), **posix**(7V), **sunos**(7), **svidiii**(7V), **xopen**(7V)

NAME

svidiii – SVIDIII lint library

SYNOPSIS

/usr/5bin/lint –n –lsvidiii *svidiii_src.c*

AVAILABILITY

This environment is not fully tested under SunOS Release 4.1 as there is no test suite available. The environment that is believed to closely approximate a SVIDIII environment is the System V environment. The System V environment is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

SVIDIII is a future environment that Sun intends to support. SunOS Release 4.1 does not currently fully support SVIDIII applications. It does support many of the functions described by the SVIDIII document. This man page does not imply that the functions supported by SunOS Release 4.1 and the functions described by the SVIDIII document perform identically. The SVIDIII lint library is intended solely as a porting aid.

The SVIDIII lint library consists exclusively of SVIDIII functions. Users may lint their code with the **–n –lsvidiii** options to catch all non-SVIDIII features.

FILES

/usr/5lib/lint/llib-lsvidiii* SVIDIII C lint library

SEE ALSO

lint(1V), **ansic**(7V), **bsd**(7), **posix**(7V), **sunos**(7), **svidii**(7V), **xopen**(7V)

NAME

xopen – overview of the X/Open Portability Guide Issue 2 (X/Open) environment

SYNOPSIS

/usr/5bin/lint –n –lxopen *xopen_src.c*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

X/Open is a set of functions and header files. The SunOS Release 4.1 implementation of X/Open is a superset — it includes all of the functionality described in the **/usr/group** Standard 1984 — as well as much of the System V functionality, and much of the SunOS functionality.

All man pages that are associated with X/Open are marked by a "V" after the section number. Not all "V" pages are X/Open, however. Some "V" pages may be part of other System V based environments such as POSIX.1.

If a user desires to work in a X/Open (or System V) environment, the user should set the path variable to include **/usr/xpg2bin** and **/usr/5bin** before anything else. The typical path is:

set path=(/usr/xpg2bin /usr/5bin /bin /usr/bin /usr/ucb)

As a portability aid, Sun is providing a lint library that consists exclusively of X/Open functions. Users may lint their code with the **–n –lxopen** options to catch all non-X/Open features.

X/Open, as with most other environments, continues to evolve. The **–lxopen** lint library will always refer to the most recent document supported by Sun. Some applications may wish to port to a particular version of the environment; they may safely use the more specific name of **–lxpg2** (currently the same as **–lxopen**).

FILES

/usr/xpg2bin/*	X/Open specific executables
/usr/xpg2include/*	X/Open specific header files
/usr/5include/*	System V specific header files
/usr/xpg2lib/*	X/Open specific library files
/usr/5lib/*	System V specific library files

SEE ALSO

lint(1V), **ansic**(7V), **bsd**(7), **posix**(7V), **sunos**(7), **svidii**(7V), **svidiii**(7V)

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance. It describes all the server processes and daemons that run on the system, as well as standalone (PROM monitor) programs.

An 8V section number means one or more of the following:

- The man page documents System V behavior only.
- The man page documents default SunOS behavior, and System V behavior as it differs from the default behavior. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents behavior compliant with *IEEE Std 1003.1-1988* (POSIX.1).

Disk formatting and labeling is done by **format**(8S). Bootstrapping of the system is described in **boot**(8S) and **init**(8). The standard set of commands run by the system when it boots is described in **rc**(8). Related commands include those that check the consistency of file systems, **fsck**(8); those that mount and unmount file systems, **mount**(8); add swap devices, **swapon**(8); force completion of outstanding file system I/O, **sync**(2); shutdown or reboot a running system **shutdown**(8), **halt**(8), and **reboot**(8); and, set the time on a machine from the time on another machine **rdate**(8C).

Creation of file systems is discussed in **mkfs**(8) and **newfs**(8). File system performance parameters can be adjusted with **tunefs**(8). File system backups and restores are described in **dump**(8) and **restore**(8).

Procedures for adding new users to a system are described in **adduser**(8), using **vipw**(8) to lock the password file during editing. **panic**(8S) which describes what happens when the system crashes, **savecore**(8) which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the **fsck**(8) file system repair program are **clri**(8), **dcheck**(8), **icheck**(8), and **ncheck**(8).

Configuring a new version of the kernel requires using the program **config**(8); major system bootstraps often require the use of **mkproto**(8). New devices are added to the **/dev** directory (once device drivers are configured into the system) using **makedev**(8) and **mknod**(8). The **installboot**(8S) command can be used to install freshly compiled programs. The **catman**(8) command preformats the on-line manual pages.

Resource accounting is enabled by the **accton** command, and summarized by **sa**(8). Login time accounting is performed by **ac**(8). Disk quotas are managed using **quot**(8), **quotacheck**(8), **quotaon**(8), and **repquota**(8).

A number of servers and daemon processes are described in this section. The **update**(8) daemon forces delayed file system I/O to occur and **cron**(8) runs periodic events (such as removing temporary files from the disk periodically). The **syslogd**(8) daemon maintains the system error log. The **init**(8) process is the initial process created when the system boots. It manages the reboot process and creates the initial login prompts on the various system terminals, using **getty**(8). The Internet super-server **inetd**(8C) invokes all other internet servers as needed. These servers include the remote shell servers **rshd**(8C) and **rexecd**(8C), the remote login server **rlogind**(8C), the FTP and TELNET daemons **ftpd**(8C), and **telnetd**(8C), the TFTP daemon **tftpd**(8C), and the mail arrival notification daemon **comsat**(8C). Other network daemons include the 'load average/who is logged in' daemon **rwhod**(8C), the routing daemon **routed**(8C), and the mail daemon **sendmail**(8).

If network protocols are being debugged, then the protocol debugging trace program **trpt**(8C) is often useful. Remote magnetic tape access is provided by **rsh** and **rmt**(8C). Remote line printer access is provided by **lpd**(8), and control over the various print queues is provided by **lpc**(8). Printer cost-accounting is done through **pac**(8).

Network host tables may be gotten from the ARPA NIC using **gettable**(8C) and converted to UNIX-system-usable format using **htable**(8).

RPC and NFS daemons

RPC and NFS daemons include:

portmap	used by RPC based services.
ypbind	used by the Network Information Service (NIS) to locate the NIS server. Note: the Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.
biod	used by NFS clients to read ahead to, and write behind from, network file systems.
nfsd	the NFS server process that responds to NFS requests on NFS server machines.
ypserv	the NIS server, typically run on each NFS server.
rstatd	the server counterpart of the remote speedometer tools.
mountd	the **mount** server that runs on NFS server machines and responds to requests by other machines to mount file systems.
rwalld	used for broadcasting messages over the network.

LIST OF MAINTENANCE COMMANDS

Name	Appears on Page	Description
ac	**ac**(8)	login accounting
acctcms	**acctcms**(8)	command summary from per-process accounting records
acctcon1	**acctcon**(8)	connect-time accounting
acctcon2	**acctcon**(8)	connect-time accounting
acctdisk	**acct**(8)	miscellaneous accounting commands
acctdusg	**acct**(8)	miscellaneous accounting commands
acctmerg	**acctmerg**(8)	merge or add total accounting files
accton	**acct**(8)	miscellaneous accounting commands
accton	**sa**(8)	system accounting
acctprc1	**acctprc**(8)	process accounting
acctprc2	**acctprc**(8)	process accounting
acctwtmp	**acct**(8)	miscellaneous accounting commands
adbgen	**adbgen**(8)	generate adb script
add_client	**add_client**(8)	create a diskless network bootable NFS client on a server
add_services	**add_services**(8)	provide software installation services for any architecture
adduser	**adduser**(8)	procedure for adding new users
adv	**adv**(8)	advertise a directory for remote access with RFS
analyze	**old-analyze**(8)	postmortem system crash analyzer
arp	**arp**(8C)	address resolution display and control
audit	**audit**(8)	audit trail maintenance
auditd	**auditd**(8)	audit daemon
audit_warn	**audit_warn**(8)	audit daemon warning script
automount	**automount**(8)	automatically mount NFS file systems
biod	**nfsd**(8)	NFS daemons
boot	**boot**(8S)	start the system kernel, or a standalone program
bootparamd	**bootparamd**(8)	boot parameter server
C2conv	**c2conv**(8)	convert system to or from C2 security
C2unconv	**c2conv**(8)	convert system to or from C2 security
captoinfo	**captoinfo**(8V)	convert a termcap description into a terminfo description
catman	**catman**(8)	create the cat files for the manual
change_login	**change_login**(8)	control screen blanking and choice of login utility
chargefee	**acctsh**(8)	shell procedures for accounting
check4	**set4**(8)	check the virtual address space limit flag in a module
chown	**chown**(8)	change owner
chroot	**chroot**(8)	change root directory for a command
chrtbl	**chrtbl**(8)	generate character classification table
ckpacct	**acctsh**(8)	shell procedures for accounting

client	**client**(8)	add or remove diskless Sun386i systems
clri	**clri**(8)	clear inode
colldef	**colldef**(8)	convert collation sequence source definition
comsat	**comsat**(8C)	biff server
config	**config**(8)	build system configuration files
copy_home	**copy_home**(8)	fetch default startup files for new home directories
crash	**crash**(8)	examine system images
cron	**cron**(8)	clock daemon
dbconfig	**dbconfig**(8)	initializes the dial box
dcheck	**dcheck**(8)	file system directory consistency check
devinfo	**devinfo**(8S)	print out system device information
devnm	**devnm**(8V)	device name
diskusg	**diskusg**(8)	generate disk accounting data by user
dkctl	**dkctl**(8)	control special disk operations
dkinfo	**dkinfo**(8)	report information about a disk's geometry and partitioning
dmesg	**dmesg**(8)	collect system diagnostic messages to form error log
dname	**dname**(8)	print RFS domain and network names
dodisk	**acctsh**(8)	shell procedures for accounting
dorfs	**dorfs**(8)	initialize, start and stop RFS automatically
dump	**dump**(8)	incremental file system dump
dumpfs	**dumpfs**(8)	dump file system information
edquota	**edquota**(8)	edit user quotas
eeprom	**eeprom**(8S)	EEPROM display and load utility
etherd	**etherd**(8C)	Ethernet statistics server
etherfind	**etherfind**(8C)	find packets on Ethernet
exportfs	**exportfs**(8)	export and unexport directories to NFS clients
extract_unbundled	**extract_unbundled**(8)	extract and execute unbundled-product installation scripts
fastboot	**fastboot**(8)	reboot/halt the system without checking the disks
fasthalt	**fastboot**(8)	reboot/halt the system without checking the disks
fingerd	**fingerd**(8C)	remote user information server
format	**format**(8S)	disk partitioning and maintenance utility
fpa_download	**fpa_download**(8)	download to the Floating Point Accelerator
fparel	**fparel**(8)	Sun FPA online reliability tests
fpaversion	**fpaversion**(8)	print FPA version, load microcode
fpurel	**fpurel**(8)	perform tests the Sun Floating Point Co-processor
fpuversion4	**fpuversion4**(8)	print the Sun-4 FPU version
fsck	**fsck**(8)	file system consistency check and interactive repair
fsirand	**fsirand**(8)	install random inode generation numbers
ftpd	**ftpd**(8C)	TCP/IP Internet File Transfer Protocol server
fumount	**fumount**(8)	force unmount of an advertised RFS resource
fusage	**fusage**(8)	RFS disk access profiler
fuser	**fuser**(8)	identify processes using a file or file structure
fwtmp	**fwtmp**(8)	manipulate connect accounting records
gettable	**gettable**(8C)	get DARPA Internet format host table from a host
getty	**getty**(8)	set terminal mode
gid_allocd	**uid_allocd**(8C)	UID and GID allocator daemons
gpconfig	**gpconfig**(8)	initialize the Graphics Processor
grpck	**grpck**(8V)	check group database entries
gxtest	**gxtest**(8S)	stand alone test for the Sun video graphics board
halt	**halt**(8)	stop the processor
hostrfs	**hostrfs**(8)	Convert IP addresses to RFS format
htable	**htable**(8)	convert DoD Internet format host table
icheck	**icheck**(8)	file system storage consistency check

idload	**idload**(8)	RFS user and group mapping
ifconfig	**ifconfig**(8C)	configure network interface parameters
imemtest	**imemtest**(8S)	stand alone memory test
in.comsat	**comsat**(8C)	biff server
inetd	**inetd**(8C)	Internet services daemon
in.fingerd	**fingerd**(8C)	remote user information server
infocmp	**infocmp**(8V)	compare or print out terminfo descriptions
in.ftpd	**ftpd**(8C)	TCP/IP Internet File Transfer Protocol server
init	**init**(8)	process control initialization
in.named	**named**(8C)	Internet domain name server
in.rexecd	**rexecd**(8C)	remote execution server
in.rlogind	**rlogind**(8C)	remote login server
in.routed	**routed**(8C)	network routing daemon
in.rshd	**rshd**(8C)	remote shell server
in.rwhod	**rwhod**(8C)	system status server
installboot	**installboot**(8S)	install bootblocks in a disk partition
install_small_kernel	**install_small_kernel**(8)	install a small, pre-configured kernel
installtxt	**installtxt**(8)	create a message archive
in.talkd	**talkd**(8C)	server for talk program
in.telnetd	**telnetd**(8C)	TCP/IP TELNET protocol server
in.tftpd	**tftpd**(8C)	TCP/IP Trivial File Transfer Protocol server
in.tnamed	**tnamed**(8C)	TCP/IP Trivial name server
intr	**intr**(8)	allow a command to be interruptible
iostat	**iostat**(8)	report I/O statistics
ipallocd	**ipallocd**(8C)	Ethernet-to-IP address allocator
kadb	**kadb**(8S)	adb-like kernel and standalone-program debugger
keyenvoy	**keyenvoy**(8C)	talk to keyserver
keyserv	**keyserv**(8C)	server for storing public and private keys
kgmon	**kgmon**(8)	generate a dump of the operating system's profile buffers
lastlogin	**acctsh**(8)	shell procedures for accounting
ldconfig	**ldconfig**(8)	link-editor configuration
link	**link**(8V)	exercise link and unlink system calls
listen	**nlsadmin**(8)	network listener service administration for RFS
lockd	**lockd**(8C)	network lock daemon
logintool	**logintool**(8)	graphic login interface
lpc	**lpc**(8)	line printer control program
lpd	**lpd**(8)	printer daemon
mailstats	**mailstats**(8)	print statistics collected by sendmail
makedbm	**makedbm**(8)	make a NIS ndbm file
MAKEDEV	**makedev**(8)	make system special files
makekey	**makekey**(8)	generate encryption key
mc68881version	**mc68881version**(8)	print the MC68881 mask number and approximate clock rate
mconnect	**mconnect**(8)	connect to SMTP mail server socket
mkfile	**mkfile**(8)	create a file
mkfs	**mkfs**(8)	construct a file system
mknod	**mknod**(8)	build special file
mkproto	**mkproto**(8)	construct a prototype file system
modload	**modload**(8)	load a module
modstat	**modstat**(8)	display status of loadable modules
modunload	**modunload**(8)	unload a module
monacct	**acctsh**(8)	shell procedures for accounting
monitor	**monitor**(8S)	system ROM monitor
mountd	**mountd**(8C)	NFS mount request server

mount	**mount**(8)	mount and unmount file systems
mount_tfs	**mount_tfs**(8)	mount and dismount TFS filesystems
named	**named**(8C)	Internet domain name server
ncheck	**ncheck**(8)	generate names from i-numbers
ndbootd	**ndbootd**(8C)	ND boot block server
netconfig	**netconfig**(8C)	PNP boot service
netstat	**netstat**(8C)	show network status
newaliases	**newaliases**(8)	rebuild the data base for the mail aliases file
newfs	**newfs**(8)	create a new file system
newkey	**newkey**(8)	create a new key in the publickey database
nfsd	**nfsd**(8)	NFS daemons
nfsstat	**nfsstat**(8C)	Network File System statistics
nlsadmin	**nlsadmin**(8)	network listener service administration for RFS
nslookup	**nslookup**(8C)	query domain name servers interactively
nsquery	**nsquery**(8)	RFS name server query
nulladm	**acctsh**(8)	shell procedures for accounting
old-analyze	**old-analyze**(8)	postmortem system crash analyzer
pac	**pac**(8)	printer/plotter accounting information
panic	**panic**(8S)	what happens when the system crashes
ping	**ping**(8C)	send ICMP ECHO_REQUEST packets to network hosts
pnpboot	**pnpboot**(8C)	pnp diskless boot service
pnpd	**pnpd**(8C)	PNP daemon
pnp.s386	**pnpboot**(8C)	pnp diskless boot service
portmap	**portmap**(8C)	TCP/IP port to RPC program number mapper
praudit	**praudit**(8)	print contents of an audit trail file
prctmp	**acctsh**(8)	shell procedures for accounting
prdaily	**acctsh**(8)	shell procedures for accounting
prtacct	**acctsh**(8)	shell procedures for accounting
pstat	**pstat**(8)	print system facts
pwck	**pwck**(8V)	check password database entries
pwdauthd	**pwdauthd**(8C)	server for authenticating passwords
quotacheck	**quotacheck**(8)	file system quota consistency checker
quotaoff	**quotaon**(8)	turn file system quotas on and off
quotaon	**quotaon**(8)	turn file system quotas on and off
quot	**quot**(8)	summarize file system ownership
rarpd	**rarpd**(8C)	TCP/IP Reverse Address Resolution Protocol server
rc	**rc**(8)	command scripts for auto-reboot and daemons
rc.boot	**rc**(8)	command scripts for auto-reboot and daemons
rc.local	**rc**(8)	command scripts for auto-reboot and daemons
rdate	**rdate**(8C)	set system date from a remote host
rdump	**dump**(8)	incremental file system dump
reboot	**reboot**(8)	restart the operating system
renice	**renice**(8)	alter nice value of running processes
repquota	**repquota**(8)	summarize quotas for a file system
restore	**restore**(8)	incremental file system restore
rexd	**rexd**(8C)	RPC-based remote execution server
rexecd	**rexecd**(8C)	remote execution server
rfadmin	**rfadmin**(8)	RFS domain administration
rfpasswd	**rfpasswd**(8)	change RFS host password
rfstart	**rfstart**(8)	start RFS
rfstop	**rfstop**(8)	stop the RFS environment
rfuadmin	**rfuadmin**(8)	RFS notification shell script
rfudaemon	**rfudaemon**(8)	Remote File Sharing daemon

rlogind	**rlogind**(8C)	remote login server
rmail	**rmail**(8C)	handle remote mail received via uucp
rm_client	**rm_client**(8)	remove an NFS client
rmntstat	**rmntstat**(8)	display RFS mounted resource information
rmt	**rmt**(8C)	remote magtape protocol module
route	**route**(8C)	manually manipulate the routing tables
routed	**routed**(8C)	network routing daemon
rpc.etherd	**etherd**(8C)	Ethernet statistics server
rpcinfo	**rpcinfo**(8C)	report RPC information
rpc.lockd	**lockd**(8C)	network lock daemon
rpc.mountd	**mountd**(8C)	NFS mount request server
rpc.rexd	**rexd**(8C)	RPC-based remote execution server
rpc.rquotad	**rquotad**(8C)	remote quota server
rpc.rstatd	**rstatd**(8C)	kernel statistics server
rpc.rusersd	**rusersd**(8C)	network username server
rpc.rwalld	**rwalld**(8C)	network rwall server
rpc.sprayd	**sprayd**(8C)	spray server
rpc.statd	**statd**(8C)	network status monitor
rpc.yppasswdd	**yppasswdd**(8C)	server for modifying NIS password file
rpc.ypupdated	**ypupdated**(8C)	server for changing NIS information
rquotad	**rquotad**(8C)	remote quota server
rrestore	**restore**(8)	incremental file system restore
rshd	**rshd**(8C)	remote shell server
rstatd	**rstatd**(8C)	kernel statistics server
runacct	**acctsh**(8)	shell procedures for accounting
runacct	**runacct**(8)	run daily accounting
rusage	**rusage**(8)	print resource usage for a command
rusersd	**rusersd**(8C)	network username server
rwalld	**rwalld**(8C)	network rwall server
rwhod	**rwhod**(8C)	system status server
sa	**sa**(8)	system accounting
savecore	**savecore**(8)	save a core dump of the operating system
sendmail	**sendmail**(8)	send mail over the internet
set4	**set4**(8)	set the virtual address space limit flag in a module
setsid	**setsid**(8V)	set process to session leader
showfhd	**showfhd**(8C)	showfh daemon run on the NFS servers
showfh	**showfh**(8C)	print full pathname of file from the NFS file handle
showmount	**showmount**(8)	show all remote mounts
shutacct	**acctsh**(8)	shell procedures for accounting
shutdown	**shutdown**(8)	close down the system at a given time
skyversion	**skyversion**(8)	print the SKYFFP board microcode version number
sprayd	**sprayd**(8C)	spray server
spray	**spray**(8C)	spray packets
start_applic	**start_applic**(8)	generic application startup procedures
startup	**acctsh**(8)	shell procedures for accounting
statd	**statd**(8C)	network status monitor
sticky	**sticky**(8)	mark files for special treatment
sundiag	**sundiag**(8)	system diagnostics
suninstall	**suninstall**(8)	install and upgrade the SunOS operating system
swapon	**swapon**(8)	specify additional device for paging and swapping
sys-config	**sys-config**(8)	configure a system or administer configuration information
syslogd	**syslogd**(8)	log system messages
sys-unconfig	**sys-unconfig**(8)	undo a system's configuration

talkd	**talkd**(8C)	server for talk program
telnetd	**telnetd**(8C)	TCP/IP TELNET protocol server
tfsd	**tfsd**(8)	TFS daemon
tftpd	**tftpd**(8C)	TCP/IP Trivial File Transfer Protocol server
tic	**tic**(8V)	terminfo compiler
tnamed	**tnamed**(8C)	TCP/IP Trivial name server
trpt	**trpt**(8C)	transliterate protocol trace
ttysoftcar	**ttysoftcar**(8)	enable/disable carrier detect
tunefs	**tunefs**(8)	tune up an existing file system
turnacct	**acctsh**(8)	shell procedures for accounting
tzsetup	**tzsetup**(8)	set up old-style time zone information in the kernel
uid_allocd	**uid_allocd**(8C)	UID and GID allocator daemons
umount	**mount**(8)	mount and unmount file systems
umount_tfs	**mount_tfs**(8)	mount and dismount TFS filesystems
unadv	**unadv**(8)	unadvertise a Remote File Sharing resource
unconfigure	**unconfigure**(8)	reset the network configuration for a Sun386i system
unlink	**link**(8V)	exercise link and unlink system calls
unset4	**set4**(8)	unset the virtual address space limit flag in a module
update	**update**(8)	periodically update the super block
user_agentd	**user_agentd**(8C)	user agent daemon
uucheck	**uucheck**(8C)	check the UUCP directories and Permissions file
uucico	**uucico**(8C)	file transport program for the UUCP system
uuclean	**uuclean**(8C)	uucp spool directory clean-up
uucleanup	**uucleanup**(8C)	UUCP spool directory clean-up
uucpd	**uucpd**(8C)	UUCP server
uusched	**uusched**(8C)	the scheduler for the UUCP file transport program
uuxqt	**uuxqt**(8C)	execute remote command requests
vipw	**vipw**(8)	edit the password file
vmstat	**vmstat**(8)	report virtual memory statistics
wtmpfix	**fwtmp**(8)	manipulate connect accounting records
ypbatchupd	**ypbatchupd**(8C)	NIS batch update daemon
ypbind	**ypserv**(8)	NIS server and binder processes
ypinit	**ypinit**(8)	build and install NIS database
ypmake	**ypmake**(8)	rebuild NIS database
yppasswdd	**yppasswdd**(8C)	server for modifying NIS password file
yppoll	**yppoll**(8)	version of NIS map at NIS server
yppush	**yppush**(8)	force propagation of changed NIS map
ypserv	**ypserv**(8)	NIS server and binder processes
ypset	**ypset**(8)	point ypbind at a particular server
ypsync	**ypsync**(8)	collect most up-to-date NIS maps
ypupdated	**ypupdated**(8C)	server for changing NIS information
ypxfr	**ypxfr**(8)	transfer NIS map from NIS server to here
zdump	**zdump**(8)	time zone dumper
zic	**zic**(8)	time zone compiler

NAME

ac – login accounting

SYNOPSIS

/usr/etc/ac [**−w** *wtmp*] [**−p**] [**−d**] [*username*] . . .

DESCRIPTION

ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file **/var/adm/wtmp** is maintained by **init**(8) and **login**(1). Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

OPTIONS

−w *wtmp*
: Specify an alternate *wtmp* file.

−p
: Print individual totals; without this option, only totals are printed.

−d
: Printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, **/var/adm/wtmp** is used.

FILES

/var/adm/wtmp

SEE ALSO

login(1), **utmp**(5V), **init**(8), **sa**(8)

NAME

acctdisk, acctdusg, accton, acctwtmp – overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**–u** *filename*] [**–p** *filename*]

/usr/lib/acct/accton [*filename*]

/usr/lib/acct/acctwtmp *reason*

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. **acctsh**(8) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/etc/utmp**, as described in **utmp**(5V). The programs described in **acctcon**(8) convert this file into session and charging records, which are then summarized by **acctmerg**(8).

Process accounting is performed by the UNIX system kernel. Upon termination of a process, one record per process is written to a file (normally **/var/adm/pacct**). The programs in **acctprc**(8) summarize this data for charging purposes; **acctcms**(8) is used to summarize command usage. Current process data may be examined using **acctcom**(1).

Process accounting and connect time accounting (or any accounting records in the format described in **acct**(5)) can be merged and summarized into total accounting records by **acctmerg** (see **tacct** format in **acct**(5)). **prtacct** (see **acctsh**(8)) is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from '**find / –print**') and computes disk resource consumption (including indirect blocks) by login.

accton without arguments turns process accounting off. If *filename* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see **acct**(2V) and **acct**(5)). You must be super-user to use this command.

acctwtmp writes a **utmp**(5V) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. The login name for this record is set to **@@acct** (see **utmp**(5V)). *reason* must be a string of 8 or fewer characters, numbers, **$**, or SPACE characters. If *reason* contains a SPACE character, it must be enclosed in double quotes. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp uname >> /var/adm/wtmp
acctwtmp fsave >> /var/adm/wtmp
```

OPTIONS

acctdusg

–u *filename*
: Place records consisting of those file names for which **acctdusg** charges no one in *filename* (a potential source for finding users trying to avoid disk charges).

–p *filename*
: Use *filename* as the password file, rather than **/etc/passwd**. (See **diskusg**(8) for more details.)

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in section 8 of this manual
/var/adm/pacct	current process accounting file
/var/adm/wtmp	login/logoff history file

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **diskusg**(8), **fwtmp**(8), **runacct**(8)

NAME

acctcms – command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [**–cjnst**] *filename* ...

/usr/lib/acct/acctcms [**–a** [**po**] [**cjnstpo**] *filename* ...

DESCRIPTION

acctcms reads one or more *filename*s, normally in the form described in **acct**(5). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format.

OPTIONS

–a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in **acctcom**(1). Output is normally sorted by total kcore-minutes.

–c Sort by total CPU time, rather than total kcore-minutes.

–j Combine all commands invoked only once under "***other".

–n Sort by number of processes.

–s Any file names encountered hereafter are already in internal summary format.

–t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both.

The following options may be used only with the **–a** option.

–p Output a prime-time-only command summary.

–o Output a non-prime (offshift) time only command summary.

When **–p** and **–o** are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non-prime.

EXAMPLES

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms -s today previoustotal >total
acctcms -a -s today
```

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **fwtmp**(8), **runacct**(8)

BUGS

Unpredictable output results if **–t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME
acctcon1, acctcon2 – connect-time accounting

SYNOPSIS
/usr/lib/acct/acctcon1 [**–pt**] [**–l** *file*] [**–o** *file*]

/usr/lib/acct/acctcon2

DESCRIPTION
acctcon1
acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/var/adm/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time.

acctcon2
acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in **acct**(5)).

OPTIONS
acctcon1
–p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

–t Test mode. **acctcon1** maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **–t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

–l *file* *file* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of **login**(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See **init**(8) and **utmp**(5V).

–o *file* *file* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

EXAMPLES
These commands are typically used as shown below. The file **ctmp** is created only for the use of **acctprc**(8) commands:

```
acctcon1 –t –l lineuse –o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES
/var/adm/wtmp

SEE ALSO
acctcom(1), **login**(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **fwtmp**(8), **init**(8), **runacct**(8)

BUGS
The line usage report is confused by date changes. Use **wtmpfix** (see **fwtmp**(8)) to correct this situation.

NAME

acctmerg – merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [**–aiptuv**] [*filename* ...]

DESCRIPTION

acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see **acct**(5)) or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys.

OPTIONS

- **–a** Produce output in ASCII version of **tacct**.
- **–i** Input files are in ASCII version of **tacct**.
- **–p** Print input with no processing.
- **–t** Produce a single record that totals all input.
- **–u** Summarize by user ID, rather than user ID and name.
- **–v** Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

acctmerg –v <filename1 >filename2
edit file2 as desired ...
acctmerg –i <filename2 >filename1

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctprc**(8), **acctsh**(8), **fwtmp**(8), **runacct**(8)

NAME

acctprc1, acctprc2 – process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [*ctmp*]

/usr/lib/acct/acctprc2

DESCRIPTION

acctprc1

acctprc1 reads input in the form described by **acct**(5), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (ticks), non-prime CPU time (ticks), and mean memory size (in pages). If *ctmp* is given, it is expected to be the name of a file containing a list of login sessions, in the form described in **acctcon**(8), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in *ctmp* helps it distinguish among different login names that share the same user ID.

acctprc2

acctprc2 reads records in the form written by **acctprc1**, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

EXAMPLES

These commands are typically used as shown below:

```
acctprc1 ctmp </var/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctsh**(8), **cron**(8), **fwtmp**(8), **runacct**(8)

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run from the command line, it is difficult to do this for those commands run by **cron**(8), for example. More precise conversion can be done by faking login sessions on the console using the **acctwtmp** program in **acct**(8).

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct – shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee *login-name number*

/usr/lib/acct/ckpacct [*blocks*]

/usr/lib/acct/dodisk [**–o**] [*filename* ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct *number*

/usr/lib/acct/nulladm *filename*

/usr/lib/acct/prctmp *filename*

/usr/lib/acct/prdaily [**–cl**] [*mmdd*]

/usr/lib/acct/prtacct *filename* [*heading*]

/usr/lib/acct/runacct [*mmdd*] [*mmdd state*]

/usr/lib/acct/shutacct [*reason*]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

chargefee

chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/var/adm/fee**, to be merged with other accounting records during the night.

ckpacct

ckpacct should be initiated by **cron**(8) every hour. It periodically checks the size of **/var/adm/pacct**. If the size exceeds *blocks*, 1000 by default, **turnacct** is called with the argument **switch**. If the number of free disk blocks in the **/usr** file system falls below 500, **ckpacct** automatically turns off the collection of process accounting records using the **off** argument to **turnacct**. When at least this number of blocks is restored, accounting is activated again. This feature is sensitive to the frequency at which **ckpacct** is executed, usually by **cron**.

dodisk

dodisk should be executed by **cron** to perform the disk accounting functions. By default, it does disk accounting on the 4.2 file systems in **/etc/fstab**. *filename*s specify the one or more filesystem names where disk accounting will be done. If *filename*s are used, disk accounting will be done on these filesystems only. They should be the special file names of mountable filesystems.

lastlogin

lastlogin is invoked by **runacct** to update **/var/adm/acct/sum/loginlog**, which shows the last date on which each person logged in. **lastlogin** deletes the entries of users no longer in **/etc/passwd** and creates new entries.

monacct

monacct should be invoked once each month or each accounting period. *number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01–12). This default is useful if **monacct** is executed by **cron**(8) on the first day of each month. **monacct** creates summary files in **/var/adm/acct/fiscal** and restarts summary files in **/var/adm/acct/sum**.

nulladm

nulladm creates *filename* with mode 664 and insures that owner and group are **adm**. It is called by various accounting shell procedures.

prctmp

prctmp can be used to print the session record file with headings (normally **/var/adm/acct/nite/ctmp** created by **acctcon1** (see **acctcon**(8)). The heading specifies device, user ID, login name, prime connect time (in seconds), non-prime connect time (in seconds), session starting time (numeric) and starting date and time.

prdaily

prdaily is invoked by **runacct** to format a report of the previous day's accounting data. The report resides in **/var/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing **prdaily**. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. Previous daily reports are cleaned up and therefore inaccessible after each invocation of **monacct**.

prtacct

prtacct can be used to format and print any total accounting (**tacct**) file with headings. See Chapter 8 in the *System and Network Administration* manual, for an explanation of this output.

runacct

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see **runacct**(8).

shutacct

shutacct should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/var/adm/wtmp**. If *reason* is not specified, **shutdown** is provided as a default reason.

startup

startup should be called by **/etc/rc** to turn the accounting on whenever the system is brought up.

turnacct

turnacct is an interface to **accton** (see **acct**(8)) to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/var/adm/pacct** to the next free name in **/var/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by **ckpacct** and thus can be taken care of by **cron** and used to keep **pacct** to a reasonable size. This command is restricted to the super-user.

OPTIONS

dodisk

−o Do a slower version of disk accounting by login directory. *filenames* should be mount points of mounted filesystem.

prdaily

−c Prints a report of exceptional resource usage by command. This may be used on current day's accounting data only.

−l Print a report of exceptional usage by login ID for the specifed date.

FILES

/etc/fstab	list of file systems **/var/adm/fee** accumulator for fees
/var/adm/pacct	current file for per-process accounting
/var/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/var/adm/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	limits for exceptional usage by login id
/usr/lib/acct/ptecms.awk	limits for exceptional usage by command name
/var/adm/acct/nite	working directory
/usr/lib/acct	directory of accounting commands
/var/adm/acct/sum	summary directory, should be saved

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **cron**(8), **diskusg**(8), **fwtmp**(8), **runacct**(8)

System and Network Administration

NAME

adbgen – generate adb script

SYNOPSIS

/usr/lib/adb/adbgen *filename*.**adb** ...

DESCRIPTION

adbgen makes it possible to write **adb**(1) scripts that do not contain hard-coded dependencies on structure member offsets. The input to **adbgen** is a file named *filename*.**adb** which contains **adbgen** header information, then a null line, then the name of a structure, and finally an **adb** script. **adbgen** only deals with one structure per file; all member names are assumed to be in this structure. The output of **adbgen** is an **adb** script in *filename*. **adbgen** operates by generating a C program which determines structure member offsets and sizes, which in turn generates the **adb** script.

The header lines, up to the null line, are copied verbatim into the generated C program. Typically these include C **#include** statements to include the header files containing the relevant structure declarations.

The **adb** script part may contain any valid **adb** commands (see **adb**(1)), and may also contain **adbgen** requests, each enclosed in {}s. Request types are:

- Print a structure member. The request form is {*member*,*format*}. *member* is a member name of the *structure* given earlier, and *format* is any valid **adb** format request. For example, to print the **p_pid** field of the *proc* structure as a decimal number, you would write **{p_pid,d}**.
- Reference a structure member. The request form is {******member*,*base*}. *member* is the member name whose value is desired, and *base* is an **adb** register name which contains the base address of the structure. For example, to get the **p_pid** field of the *proc* structure, you would get the proc structure address in an **adb** register, say <f, and write **{*p_pid,<f}**.
- Tell **adbgen** that the offset is ok. The request form is **{OFFSETOK}**. This is useful after invoking another **adb** script which moves the **adb** *dot*.
- Get the size of the *structure*. The request form is **{SIZEOF}**. **adbgen** replaces this request with the size of the structure. This is useful in incrementing a pointer to step through an array of structures.
- Get the offset to the end of the structure. The request form is **{END}**. This is useful at the end of the structure to get **adb** to align the **dot** for printing the next structure member.

adbgen keeps track of the movement of the **adb** *dot* and emits **adb** code to move forward or backward as necessary before printing any structure member in a script. **adbgen**'s model of the behavior of **adb**'s **dot** is simple: it is assumed that the first line of the script is of the form *struct_address*/*adb text* and that subsequent lines are of the form +/*adb text*. This causes the *adb* **dot** to move in a sane fashion. **adbgen** does not check the script to ensure that these limitations are met. **adbgen** also checks the size of the structure member against the size of the **adb** format code and warns you if they are not equal.

EXAMPLE

If there were an include file **x.h** which contained:

```
struct x {
        char    *x_cp;
        char    x_c;
        int     x_i;
};
```

Then an **adbgen** file (call it **script.adb**) to print it would be:

```
#include "x.h"
x
./"x_cp"16t"x_c"8t"x_i"n{x_cp,X}{x_c,C}{x_i,D}
```

After running **adbgen** the output file **script** would contain:

16t"x_c"8t"x_i"nXC+D"" ./"x_cp"16t"x_c"8t"x_i"nXC+D

To invoke the script you would type:

x$<script

FILES

/usr/lib/adb/* **adb** scripts for debugging the kernel

SEE ALSO

adb(1), **kadb**(8S)

Debugging Tools

BUGS

adb syntax is ugly; there should be a higher level interface for generating scripts.

Structure members which are bit fields cannot be handled because C will not give the address of a bit field. The address is needed to determine the offset.

DIAGNOSTICS

Warnings about structure member sizes not equal to **adb** format items and complaints about badly formatted requests. The C compiler complains if you reference a structure member that does not exist. It also complains about & before array names; these complaints may be ignored.

NAME

add_client – create a diskless network bootable NFS client on a server

SYNOPSIS

/usr/etc/install/add_client [**–inpv**] [**–a** *kernel-arch*] [**–e** *exec-path*] [**–f** *share-path*] [**–h** *home-path*] [**–k** *kvm-path*] [**–m** *mail-path*] [**–r** *root-path*] [**–s** *swap-path*] [**–t** *term-type*] [**–y** *yptype*] [**–z** *swapsize*] [*client* . . .]

DESCRIPTION

add_client adds an NFS client to a server. It can only be run by the super-user.

A default standard layout is used to set up the client's environment, but most pathnames can be overridden with the appropriate option, or menu field change.

Before you can add a client, you must first make sure that the Internet and Ethernet addresses for *client* are listed in the Network Interface Service (NIS) hosts database (if the server is running the NIS service), or in the server's **/etc/hosts** and **/etc/ethers** databases, respectively. If **add_client** cannot find the client entry in the hosts database it aborts the operation. If there is no client entry in the **/etc/ethers** database, **add_client** issues a warning to update this file while adding the client.

The default root and swap partitions are **/export/root/***client* and **/export/swap/***client*, respectively.

add_client updates the **/etc/bootparams** file on the server but not the bootparams database in the NIS service (if used).

If the server is not running as an NIS master, **add_client** issues a warning to indicate that the database is out of date and the NIS master should be updated.

add_client updates the server's **/etc/exports** file to allow client's root access to each client's root file system. It also exports each client's swap file accordingly. Note: the system administrator should verify that the **/etc/exports** file contains correct information, and that file systems are exported to the correct users and groups. Refer to **exportfs**(8) for details on exporting file systems.

If the **–i** or **–p** option is not specified, at least one *client* argument must be supplied on the command line.

OPTIONS

–i Interactive. Bring up a full-screen menu interface to **add_client**.

–n Print the working parameters and exit without doing anything. This is used to verify what parameters **add_client** will use before actually doing anything.

–p Display a short version of all client information, If *client*s are specified on the command line, only display information for those clients. When combined with the **–v** option, a long version of client information is displayed.

–v Verbose. Report information about the client as steps are performed.

–a *kernel-arch* Specify the client kernel architecture (for instance, **sun3**, **sun4**, **sun4c**. . .). **add_client** prompts for the kernel architecture when unable to determine the correct value.

–e *exec-path* Set the pathname of the directory in which the executables for the architecture specified by **–a**. The client mounts **/export/exec/***arch.rel* as **/usr**. See WARNINGS.

–f *share-path* Set the pathname of the share directory, which is normally a link to **/usr/share**.

–h *home-path* Set the pathname of the directory for the client's home. The default is **/home/***server-name*.

–k *kvm-path* Set the pathname of the directory containing the client's kernel executables. See WARNINGS.

–m *mail-path* Set the pathname of the client's mail directory. The default is **/var/spool/mail**.

–r *root-path* Set the pathname of parent directory for client root directories; *root/client* is the pathname of the client's root directory. The default is **/export/root/***client-name*.

−s *swap-path* Set the pathname of parent directory for client swap files; *swap/client* is the pathname of the client's swap file. The default is **/export/swap/***client-name*.

−t *term-type* Set the terminal type of the client's console.

−y *yptype* Indicate the type of NIS server or if client is to be an NIS client; it can be **client** or **none**. The **none** argument results in the NIS service being disabled on the client. The default is **client**.

−z *swapsize* Reserve *swapsize* bytes for the client's swap file. **swapsize** can be flagged as kilobytes, blocks, or megabytes, with the **k**, **b**, or **m** suffixes, respectively. The default is 16Mb, and bytes are used when no units are specified.

FILES

/etc/bootparams
/etc/ethers
/etc/exports
/etc/hosts
/export/exec/proto.root.*release*
architecture independent base for the client root file system
/tftpboot.*client-ipaddr* link to **/tftpboot/boot.***arch*

SEE ALSO

add_services(8), **bootparamd(8), exportfs**(8), **ndbootd**(8C), **rm_client**(8), **suninstall**(8)

Installing SunOS 4.1

DIAGNOSTICS

add_client: must be super-user
You must be root to use **add_client**.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

WARNINGS

The **−e** *exec-path* and the **−k** *kvm-path* options should not be used since the correct paths are determined when the adding the client's architecture service. See **add_services**(8).

NAME

add_services – provide software installation services for any architecture

SYNOPSIS

/usr/etc/install/add_services

DESCRIPTION

add_services is a menu-based program to setup a system as a server and/or to add additional software categories or other architecture releases. It is used to provide support to diskless clients, dataless clients, or just to act as a file server. **add_services** can only be run by the super-user.

add_services updates the **/etc/exports** file (see **exports**(5) and **exportfs**(8)) to export the necessary file systems to become a file server. After running **add_services**, the system administrator should verify this file to make sure that the new services have been exported to the correct groups.

FILES

/etc/hosts	hosts database, host must be in this database or in the Network Interface Service (NIS) hosts map
/etc/exports	database of exported file systems, service related directories must be exported
/tftpboot	add_services sets up this directory in order to provide boot service to clients

SEE ALSO

exports(5), **add_client**(8), **exportfs**(8), **rm_client**(8), **suninstall**(8)

Installing SunOS 4.1

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

adduser – procedure for adding new users

DESCRIPTION

To add an account for a new user, the system administrator (or super-user):

- Create an entry for the new user in the system password files.
- Create a home directory for the user, and change ownership so the new user owns that directory.
- Optionally set up skeletal dot files for the new user (**.cshrc**, **.login**, **.profile**...).
- If the account is on a system running the Network Interface Service (NIS), take additional measures.

USAGE

Making an Entry in the Password File

To add an entry for the new login name on a local host, first edit the **/etc/passwd** file — inserting a line for the new user. This must be done with the password file locked, for instance, by using **vipw**(8), and the insertion must be made above the line containing the string:

+::0:0:::

This line indicates that additional accounts can be found in the NIS service.

To add an entry for the new login name into the NIS service, add an identical line to the file **/etc/passwd** on the NIS master server, and run **make**(1) in the directory **/var/yp** (see **ypmake**(8) for details) to propagate the change.

The new user is assigned a group and user ID number (GID and UID respectively). UIDs should be unique for each user and consistent across the NFS domain, since they control access to files. GIDs need not be unique. Typically, users working on similar projects will assigned to the same group. The system staff is group 10 for historical reasons, and the super-user is in this group.

An entry for a new user **francine** would look like this:

francine::235:20:& Featherstonehaugh:/usr/francine:/bin/csh

Fields in each password-file entry are delimited by colons, and have the following meanings:

- Login name (**francine**). The login name is limited to eight characters in length.
- Encrypted password or the string *##name* if encrypted passwords are stored in the password adjunct file. Typically, if passwords are to be stored in the main password file, this field is left empty, so no password is needed when the user first logs in. If security demands a password, it should be assigned by running **passwd**(1) immediately after exiting the editor. The number of significant characters in a password is eight. (See **passwd**(1).)
- User ID. The UID is a number which identifies that user uniquely in the system. Files owned by the user have this number stored in their data blocks, and commands such as **ls (1V)** (see **ls**(1V)), use it to look up the owner's login name. For this reason, you cannot randomly change this number. See **passwd**(5) for more information.
- Group ID. The GID number identifies the group to which the user belongs by default (although the user may belong to additional groups as well). All files that the user creates have this number stored in their data blocks, and commands such as **ls**(1V) (see **ls**(1V), use it to look up the group name. Group names and assignments are listed in the file **/etc/group** (which is described in **group**(5)) or in the NIS group map.
- This field is called the GCOS field (from earlier implementation of the operating system) and is traditionally used to hold the user's full name. Some installations have other information encoded in this field. From this information we can tell that Francine's real name is 'Francine Featherstonehaugh'. The **&** in the entry is shorthand for the user's login name.

- User's home directory. This is the directory in which that user is "positioned" when they log in.
- Initial shell which this user will see on login. If this field is empty, **sh**(1) is used as the initial shell.

An entry for a new user **francine** would look like this:

francine:::::lo:ad,+dw

Fields in each password adjunct file entry are delimited by colons, and have the following meanings:

- Login name (**francine**). This name must match the login name in the password file.
- Encrypted password. Typically, this field is left empty when adding the line using the editor. **passwd**(1) should be run immediately after exiting the editor.
- The next three fields are the minimum label, the maximum label, and the default label. These fields should be left empty, since they are reserved for future use.
- The next two fields are for the always-audit flags and the never-audit flags. Always-audit flags specify which events are guaranteed to be audited for that user. Never-audit flags specify which events are guaranteed not to be audited for that user. For a description of audit flags, see **audit_data**(5).

Making a Home Directory

As shown in the password file entry above, the name of Francine's home directory is to be **/usr/francine**. This directory must be created using **mkdir**(1), and Francine must be given ownership of it using **chown**(8), in order for her profile files to be read and executed, and to have control over access to it by other users:

```
example# mkdir /usr/francine
example# /usr/etc/chown francine /usr/francine
```

If running under NFS, the **mkdir**(1) and **chown**(8) commands must be performed on the NFS server.

Setting Up Skeletal Profile Files

New users often need assistance in setting up their profile files to initialize the terminal properly, configure their search path, and perform other desired functions at startup. Providing them with skeletal profile files saves time and interruptions for both the new user and the system administrator.

Such files as **.profile** (if they use **/usr/bin/sh** as the shell), or **.cshrc** and **.login** (if they use **/usr/bin/csh** as the shell), can include commands that are performed automatically at each login, or whenever a shell is invoked, such as **tset**(1). The ownership of these files must be changed to belong to the new user, either by running **su**(1V) before making copies, or by using **chown**(8).

FILES

/etc/passwd	password file
/etc/security/passwd.adjunct	
/etc/group	group file
/etc/yp/src/passwd	
~/.cshrc	
~/.login	
~/.profile	

SEE ALSO

csh(1), **ls**(1V), **make**(1), **mkdir**(1), **passwd**(1), **sh**(1), **su**(1V), **tset**(1), **audit**(2), **audit_control**(5), **audit_data**(5), **passwd.adjunct**(5), **group**(5), **passwd**(5), **passwd.adjunct**(5) **audit**(8), **auditd**(8), **chown**(8), **vipw**(8), **ypmake**(8)

System and Network Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

adv – advertise a directory for remote access with RFS

SYNOPSIS

adv
adv [**–r**] [**–d** *description*] *resource pathname* [*clients* . . .]
adv –m *resource* **–d** *description* | [*clients*. . .]
adv –m *resource* [**–d** *description*] | *clients* . . .

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

adv makes a resource from one system available for use on other systems. The machine that advertises the resource is called the server, while systems that mount and use the resource are **clients**. See **mount**(8). *resource* represents a directory, which could contain files, subdirectories, named pipes and devices.

Remote File Sharing (RFS) must be running before **adv** can be used to advertise or modify a resource entry.

When used with no options, **adv** displays all local resources that have been advertised; this includes the resource name, the pathname, the description, the read-write status, and the list of authorized clients. The resource field has a fixed length of 14 characters; all others are of variable length. Fields are separated by two SPACE characters and double quotes (") surround the description.

This command may be used without options by any user; otherwise it is restricted to the super-user.

There are three ways **adv** is used:

- To print a list of all locally-advertised resources, as shown by the first synopsis.
- To advertise the directory *pathname* under the name *resource* so it is available to RFS *clients*, as shown by the second synopsis.
- To modify *client* and *description* fields for currently advertised resources, as shown by the third and fourth synopses.

If any of the following are true, an error message will be sent to standard error.

- The network is not up and running.
- *pathname* is not a directory.
- *pathname* is not on a file system mounted locally.
- There is at least one entry in the *clients* field but none are syntactically valid.

OPTIONS

–r Restrict access to the resource to a read-only basis. The default is read-write access.

–d *description* Provide brief textual information about the advertised resource. *description* is a single argument surrounded by double quotes ("*argument*") and has a maximum length of 32 characters.

–m *resource* Modify information for a resource that has already been advertised. The resource is identified by a *resource* name. Only the *clients* and *description* fields can be modified. To change the *pathname*, *resource* name, or read/write permissions, you must unadvertise and re-advertise the resource.

resource This is the symbolic name used by the server and all authorized clients to identify the resource. It is limited to a maximum of 14 characters and must be different from every other resource name in the domain. All characters must be printable ASCII characters, but must not include '.' (periods), '/' (slashes), or white space.

pathname This is the local pathname of the advertised resource. It is limited to a maximum of 64 characters. This pathname cannot be the mount point of a remote resource and it can only be advertised under one resource name.

clients These are the names of all clients that are authorized to remotely mount the resource. The default is that all machines that can connect to the server are authorized to access the resource. Valid input is of the form *nodename*, *domain.nodename*, *domain.*, or an alias that represents a list of client names. A domain name must be followed by a '.' to distinguish it from a host name. The aliases are defined in **/etc/host.alias** and must conform to the alias capability in **mail**(1).

EXAMPLES

The following example displays the local resources that have been advertised:

```
example% adv
LOCAL_SUN3    /export/local/sun3  ""  read-only unrestricted
LOCAL_SUN4    /export/local/sun4  ""  read-only unrestricted
LOCAL_SHARE   /export/local/share  ""  read-only unrestricted
```

EXIT STATUS

If there is at least one syntactically valid entry in the *clients* field, a warning will be issued for each invalid entry and the command will return a successful exit status. A non-zero exit status will be returned if the command fails.

FILES

/etc/host.alias

SEE ALSO

mount(8), **rfstart**(8), **unadv**(8)

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*

arp –a [*vmunix* [*kmem*]]

arp –d *hostname*

arp –s *hostname ether_address* [**temp**] [**pub**] [**trail**]

arp –f *filename*

DESCRIPTION

The **arp** program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (**arp**(4P)).

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation.

OPTIONS

–a Display all of the current ARP entries by reading the table from the file *kmem* (default **/dev/kmem**) based on the kernel file *vmunix* (default **/vmunix**).

–d Delete an entry for the host called *hostname*. This option may only be used by the super-user.

–s Create an ARP entry for the host called *hostname* with the Ethernet address *ether_address*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be published, for instance, this system will respond to ARP requests for *hostname* even though the hostname is not its own. The word **trail** indicates that trailer encapsulations may be sent to this host.

–f Read the file named *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_address [**temp**] [**pub**] [**trail**]

with argument meanings as given above.

SEE ALSO

arp(4P), **ifconfig**(8C)

NAME

audit – audit trail maintenance

SYNOPSIS

audit [**–n** | **–s** | **–t**]
audit –d *username*
audit –u *username audit_event_state*

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

The **audit** command is the general administrator's interface to kernel auditing. The process audit state for a user can be temporarily or permanently altered. The audit daemon may be notified to read the contents of the **audit_control** file and re-initialize the current audit directory to the first directory listed in the **audit_control** file, or to open a new audit file in the current audit directory specified in the **audit_control** file as last read by the audit daemon. Auditing may also be terminated/disabled.

OPTIONS

–n Signal audit daemon to close the current audit file and open a new audit file in the current audit directory.

–s Signal audit daemon to read audit control file. The audit daemon stores the information internally.

–t Signal audit daemon to disable auditing and die.

–d *username*
Change the process audit state of all processes owned by *username*. This new process audit state is constructed from the system and user audit values as specified in the **audit_control** and **passwd.adjunct** files respectively.

–u *username audit_event_state*
Set the process audit state from *audit_event_state* for all current processes owned by *username*. See **audit_control**(5) for the format of the system audit value. The process audit state is one argument. Enclose the audit event state in quotes, or do not use SPACE characters in the process audit state specification. A new login session reconstructs the process audit state from the audit flags in the **audit_control** and **passwd.adjunct** files.

SEE ALSO

audit(2), **setuseraudit**(2), **getauditflags**(3), **getfauditflags**(3), **audit_control**(5), **passwd.adjunct**(5)

NAME

auditd – audit daemon

SYNOPSIS

/usr/etc/auditd

DESCRIPTION

The audit daemon controls the generation and location of audit trail files. If the function **issecure**(3) returns false, the only action that **auditd** takes is to disable the auditing system; otherwise, auditing is set up and started. If auditing is desired, **auditd** reads the **audit_control**(5) file to get a list of directories into which audit files can be written and the percentage limit for how much space to reserve on each filesystem before changing to the next directory.

If **auditd** receives the signal **SIGUSR1**, the current audit file is closed and another is opened. If **SIGHUP** is received, the current audit trail is closed, the **audit_control** file reread, and a new trail is opened. If **SIGTERM** is received, the audit trail is closed and auditing is terminated. The program **audit**(8) sends these signals and is recommended for this purpose.

Each time the audit daemon opens a new audit trail file, it updates the file **audit_data**(5) to include the correct name.

Auditing Conditions

The audit daemon invokes the program **audit_warn**(8) under the following conditions with the indicated options:

audit_warn soft *pathname*

The file system upon which *pathname* resides has exceeded the minimum free space limit defined in **audit_control**(5). A new audit trail has been opened on another file system.

audit_warn allsoft

All available file systems have been filled beyond the minimum free space limit. A new audit trail has been opened anyway.

audit_warn hard *pathname*

The file system upon which *pathname* resides has filled or for some reason become unavailable. A new audit trail has been opened on another file system.

audit_warn allhard *count*

All available file systems have been filled or for some reason become unavailable. The audit daemon will repeat this call to **audit_warn** every twenty seconds until space becomes available. *count* is the number of times that **audit_warn** has been called since the problem arose.

audit_warn ebusy

There is already an audit daemon running.

audit_warn tmpfile

The file **/etc/security/audit/audit_tmp** exists, indicating a fatal error.

audit_warn nostart

The internal system audit condition is AUC_FCHDONE. Auditing cannot be started without rebooting the system.

audit_warn auditoff

The internal system audit condition has been changed to not be AUC_AUDITING by someone other than the audit daemon. This causes the audit daemon to exit.

audit_warn postsigterm

An error occurred during the orderly shutdown of the auditing system.

audit_warn getacdir

There is a problem getting the directory list from **/etc/security/audit/audit_control**.

The audit daemon will hang in a sleep loop until this file is fixed.

FILES

/etc/security/audit/audit_control
/etc/security/audit/audit_data

SEE ALSO

auditsvc(2), audit_control(5), audit.log(5), audit(8), audit_warn(8)

NAME

audit_warn – audit daemon warning script

SYNOPSIS

/usr/etc/audit_warn [*option* [*arguments*]]

DESCRIPTION

The **audit_warn** script processes warning or error messages from the audit daemon. When a problem is encountered, the audit daemon, **auditd**(8) calls **audit_warn** with the appropriate arguments. The *option* argument specifies the error type.

The system administrator can specify a list of mail recpients using the script's **RECIPIENTS** variable. The default recipient is root.

OPTIONS

soft *filename*
: indicates that the soft limit for *filename* has been exceeded. The default action for this option is to send mail to the system administrator.

allsoft
: indicates that the soft limit for all filesystems has been exceeded. The default action for this option is to send mail to the system administrator.

hard *filename*
: indicates that the hard limit for the file has been exceeded. The default action for this option is to send mail to the system administrator.

allhard *count*
: indicates that the hard limit for all filesystems has been exceeded *count* times. The default action for this option is to send mail to the system administrator only if the *count* is **1**, and to send a message to console every time. It is recommended that mail *not* be send every time.

ebusy
: indicates that the audit daemon is already running. The default action for this option is to send mail to the system administrator.

tmpfile
: indicates that the temporary audit file already exists indicating a fatal error. The default action for this option is to send mail to the system administrator.

nostart
: indicates that auditing cannot be started because the system audit state is **AUC_FCHDONE**. The default action for this option is to send mail to the system administrator. Some system administrators may prefer to have the script reboot the system at this point.

auditoff
: indicates that someone other than the audit daemon changed the system audit state to something other than **AUC_AUDITING**. The audit daemon will have exited in this case. The default action for this option is to send mail to the system administrator.

postsigterm
: indicates that an error occurred during the orderly shutdown of the audit daemon. The default action for this option is to send mail to the system administrator.

getacdir
: indicates that there is a problem getting the directory list from: **/etc/security/audit/audit_control**.

  The audit daemon will hang in a sleep loop until the file is fixed.

SEE ALSO

audit.log(5), **audit_control**(5), **audit**(8), **auditd**(8)

NAME

automount – automatically mount NFS file systems

SYNOPSIS

automount [**–mnTv**] [**–D** *name= value*] [**–f** *master-file*] [**–M** *mount-directory*] [**–tl** *duration*] [**–tm** *interval*] [**–tw** *interval*] [*directory map* [*–mount-options*]] ...

DESCRIPTION

automount is a daemon that automatically and transparently mounts an NFS file system as needed. It monitors attempts to access directories that are associated with an **automount** map, along with any directories or files that reside under them. When a file is to be accessed, the daemon mounts the appropriate NFS file system. You can assign a map to a directory using an entry in a direct **automount** map, or by specifying an indirect map on the command line.

The **automount** daemon appears to be an NFS server to the kernel. **automount** uses the map to locate an appropriate NFS file server, exported file system, and mount options. It then mounts the file system in a temporary location, and creates a symbolic link to the temporary location. If the file system is not accessed within an appropriate interval (five minutes by default), the daemon unmounts the file system and removes the symbolic link. If the indicated directory has not already been created, the daemon creates it, and then removes it upon exiting.

Since the name-to-location binding is dynamic, updates to an **automount** map are transparent to the user. This obviates the need to "pre-mount" shared file systems for applications that have "hard coded" references to files.

If the *directory* argument is a pathname, the *map* argument must be an *indirect* map. In an indirect map the key for each entry is a simple name that represents a symbolic link within *directory* to an NFS mount point.

If the *directory* argument is '/–', the map that follows must be a *direct* map. A direct map is not associated with a single directory. Instead, the key for each entry is a full pathname that will itself appear to be a symbolic link to an NFS mount point.

A map can be a file or a Network Interface Service (NIS) map; if a file, the *map* argument must be a full pathname.

The *–mount-options* argument, when supplied, is a comma-separated list of **mount**(8) options, preceded by a '–'. If these options are supplied, they become the default mount options for all entries in the map. Mount options provided within a map entry override these defaults.

OPTIONS

–m Suppress initialization of *directory-map* pairs listed in the **auto.master** NIS database.

–n Disable dynamic mounts. With this option, references through the **automount** daemon only succeed when the target filesystem has been previously mounted. This can be used to prevent NFS servers from cross-mounting each other.

–T Trace. Expand each NFS call and display it on the standard output.

–v Verbose. Log status and/or warning messages to the console.

–D *envar=value*
: Assign *value* to the indicated **automount** (environment) variable.

–f *master-file*
: Read a local file for initialization, ahead of the **auto.master** NIS map.

–M *mount-directory*
: Mount temporary file systems in the named directory, instead of **/tmp_mnt**.

–tl *duration*
: Specify a *duration*, in seconds, that a file system is to remain mounted when not in use. The default is 5 minutes.

−tm *interval*
Specify an *interval*, in seconds, between attempts to mount a filesystem. The default is 30 seconds.

−tw *interval*
Specify an *interval*, in seconds, between attempts to unmount filesystems that have exceeded their cached times. The default is 1 minute.

ENVIRONMENT

Environment variables can be used within an **automount** map. For instance, if **$HOME** appeared within a map, **automount** would expand it to its current value for the **HOME** variable. Environment variables are expanded only for the automounter's environment — not for the environment of a user using the automounter's services.

The special reference to **$ARCH** expands to the output of **arch (1).** This can be useful in creating a map entry for mounting executables using a server's export pathname that varies according to the architecture of the client reading the map.

If a reference needs to be protected from affixed characters, you can surround the variable name with curly braces.

USAGE

Map Entry Format

A simple map entry (mapping) takes the form:

key [*−mount-options*] *location* ...

where *key* is the full pathname of the directory to mount when used in a direct map, or simple name in an indirect map. *mount-options* is a comma-separated list of **mount** options, and *location* specifies a remote filesystem from which the directory may be mounted. In the simple case, *location* takes the form:

hostname:*pathname*

Replicated Filesystems

Multiple *location* fields can be specified for replicated read-only filesystems, in which case **automount** sends multiple **mount** requests; **automount** mounts the file system from the first host that replies to the **mount** request. This request is first made to the local net or subnet. If there is no response, any connected server may respond. Since **automount** does not monitor the status of the server while the filesystem is mounted it will not use another location in the list if the currently mounted server crashes. This support for replicated filesystems is available only at mount time.

If each *location* in the list shares the same *pathname* then a single *location* may be used with a comma-separated list of hostnames.

hostname,*hostname* ... : *pathname*

Sharing Mounts

If *location* is specified in the form:

hostname:*pathname*:*subdir*

hostname is the name of the server from which to mount the file system, *pathname* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts when multiple directories in the same remote file system may be accessed. With a map for **/home** such as:

able **homeboy:/home/homeboy:able**
baker **homeboy:/home/homeboy:baker**

and a user attempting to access a file in **/home/able**, **automount** mounts **homeboy:/home/homeboy**, but creates a symbolic link called **/home/able** to the **able** subdirectory in the temporarily-mounted filesystem. If a user immediately tries to access a file in **/home/baker**, **automount** needs only to create a symbolic link that points to the **baker** subdirectory; **/home/homeboy** is already mounted.

With the following map:

able **homeboy:/home/homeboy/able**
baker **homeboy:/home/homeboy/baker**

automount would have to mount the filesystem twice.

Comments and Quoting

A mapping can be continued across input lines by escaping the NEWLINE with a backslash. Comments begin with a # and end at the subsequent NEWLINE.

Characters that have special significance to the **automount** map parser may be protected either with double quotes (") or by escaping with a backslash (\). Pathnames with embedded whitespace, colons (:) or dollar ($) should be protected.

Directory Pattern Matching

The '**&**' character is expanded to the value of the *key* field for the entry in which it occurs. In this case:

able **homeboy:/home/homeboy:&**

the **&** expands to **able**.

The '*' character, when supplied as the *key* field, is recognized as the catch-all entry. Such an entry will be used if any previous entry has not successfully matched the key being searched for. For instance, if the following entry appeared in the indirect map for **/home**:

* **&:/home/&**

this would allow automatic mounts in **/home** of any remote file system whose location could be specified as:

hostname:**/home/***hostname*

Multiple Mounts

A multiple mount entry takes the form:

key [/[*mountpoint* [*−mount-options*] *location* ...] ...

The initial / within the '/[*mountpoint*]' is required; the optional *mountpoint* is taken as a pathname relative to the destination of the symbolic link for *key*. If *mountpoint* is omitted in the first occurrence, a *mountpoint* of / is implied.

Given the direct map entry:

```
/arch/src  \
        /            -ro,intr   arch:/arch/src            alt:/arch/src  \
        /1.0         -ro,intr   alt:/arch/src/1.0         arch:/arch/src/1.0  \
        /1.0/man     -ro,intr   arch:/arch/src/1.0/man    alt:/arch/src/1.0/man
```

automount would automatically mount **/arch/src**, **/arch/src/1.0** and **/arch/src/1.0/man**, as needed, from either **arch** or **alt**, whichever host responded first. If the mounts are hierarchically related mounts closer to the root must appear before submounts. All the mounts of a multiple mount entry will occur together and will be unmounted together. This is important if the filesystems reference each other with relative symbolic links. Multiple mount entries can be used both in direct maps and in indirect maps.

Included Maps

The contents of another map can be included within a map with an entry of the form:

+mapname

mapname can either be a filename, or the name of an NIS map, or one of the special maps described below. If the key being searched for is not located in an included map, the search continues with the next entry.

Special Maps

There are two special maps currently available: **–hosts**, and **–null**. The **–hosts** map uses the NIS **hosts.byname** map to locate a remote host when the hostname is specified. This map specifies mounts of all exported file systems from any host. For instance, if the following **automount** command is already in effect:

automount /net –hosts

then a reference to **/net/hermes/usr** would initiate an automatic mount of all file systems from **hermes** that **automount** can mount; references to a directory under **/net/hermes** will refer to the corresponding directory relative to **hermes** root.

The **–null** map, when indicated on the command line, cancels any subsequent map for the directory indicated. It can be used to cancel a map given in **auto.master** or for a mount point specified as an entry in a direct map.

Configuration and the auto.master Map

automount normally consults the **auto.master** NIS configuration map for a list of initial **automount** maps, and sets up automatic mounts for them in addition to those given on the command line. If there are duplications, the command-line arguments take precedence over a local **–f** master map and they both take precedence over an NIS **auto.master** map. This configuration database contains arguments to the **automount** command, rather than mappings; unless **–f** is in effect, **automount** does *not* look for an **auto.master** file on the local host.

Maps given on the command line, or those given in a local **auto.master** file specified with **–f** override those in the NIS **auto.master** map. For instance, given the command:

automount –f /etc/auto.master /home –null /– /etc/auto.direct

and a file named **/etc/auto.master** that contains:

/home auto.home

automount would ignore /home entry in /etc/auto.master.

FILES

/tmp_mnt directory under which filesystems are dynamically mounted

SEE ALSO

df(1V), **ls**(1V), **stat**(2V), **passwd**(5), **mount**(8)

System and Network Administration

NOTES

The **–hosts** map must mount all the exported filesystems from a server. If frequent access to just a single filesystem is required it is more efficient to access the filesystem with a map entry that is tailored to mount just the filesystem of interest.

When it receives signal number 1, **SIGHUP**, **automount** rereads the **/etc/mtab** file to update its internal record of currently-mounted file systems. If a file system mounted with **automount** is unmounted by a **umount** command, **automount** should be forced to reread the file.

An **ls**(1V) listing of the entries in the directory for an indirect map shows only the symbolic links for currently mounted filesystems. This restriction is intended to avoid unnecessary mounts as a side effect of programs that read the directory and **stat**(2V) each of the names.

Mount points for a single automounter must not be hierarchically related. **automount** will not allow an automount mount point to be created within an automounted filesystem.

automount must not be terminated with the SIGKILL signal (**kill –9**). Without an opportunity to unmount itself, the **automount** mount points will appear to the kernel to belong to a non-responding NFS server. The recommended way to terminate **automount** services is to send a **SIGTERM** (**kill –15**) signal to the daemon. This allows the automounter to catch the signal and unmount not only its daemon but also any mounts in **/tmp_mnt**. Mounts in **/tmp_mnt** that are busy will not be unmounted.

Since each direct map entry results in a separate mount for the mount daemon such maps should be kept short. Entries added to a direct map will have no effect until the automounter is restarted.

Entries in both direct and indirect maps can be modified at any time. The new information will be used when **automount** next uses the map entry to do a mount. **automount** does not cache map entries.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

BUGS

The **bg** mount option is not recognized by the automounter.

Since **automount** is single-threaded, any request that is delayed by a slow or non-responding NFS server will delay all subsequent automatic mount requests until it completes.

Programs that read **/etc/mtab** and then touch files that reside under automatic mount points will introduce further entries to the file.

Automatically-mounted file systems are mounted with type **ignore**; they do not appear in the output of either **mount**(8), or **df**(1V).

NAME

boot – start the system kernel, or a standalone program

SYNOPSIS

>b [*device* [(*c*, *u*, *p*)]] [*filename*] [**–av**] *boot-flags*
>b?
>b!

DESCRIPTION

The boot program is started by the PROM monitor and loads the kernel, or another executable program, into memory.

The form **b?** displays all boot devices and their device arguments.

The form **b!** boots, but does not perform a RESET.

USAGE

Booting Standalone

When booting standalone, the boot program (**/boot**) is brought in by the PROM from the file system. This program contains drivers for all devices.

Booting a Sun-3 System Over the Network

When booting over the network, the Sun-3 system PROM obtains a version of the boot program from a server using the Trivial File Transfer Protocol (TFTP). The client broadcasts a RARP request containing its Ethernet address. A server responds with the client's Internet address. The client then sends a TFTP request for its boot program to that server (or if that fails, it broadcasts the request). The filename requested (unqualified — not a pathname) is the hexadecimal, uppercase representation of the client's Internet address, for example:

Using IP Address 192.9.1.17 = C0090111

When the Sun server receives the request, it looks in the directory **/tftpboot** for *filename*. That file is typically a symbolic link to the client's boot program, normally **boot.sun3** in the same directory. The server invokes the TFTP server, **tftpd**(8C), to transfer the file to the client.

When the file is successfully read in by the client, the boot program jumps to the load-point and loads **vmunix** (or a standalone program). In order to do this, the boot program makes a broadcast RARP request to find the client's IP address, and then makes a second broadcast request to a **bootparamd**(8) bootparams daemon, for information necessary to boot the client. The bootparams daemon obtains this information either from a local **/etc/bootparams** database file, or from a Network Interface Service (NIS) map. The boot program sends two requests to the bootparams daemon, the first, **whoami**, to obtain its hostname, and the second, **getfile**, to obtain the name of the client's server and the pathname of the client's root partition.

The boot program then performs a **mount**(8) operation to mount the client's root partition, after which it can read in and execute any program within that partition by pathname (including a symbolic link to another file within that same partition). Typically, it reads in the file **/vmunix**. If the program is not read in successfully, **boot** responds with a short diagnostic message.

Booting a Sun-2, Sun-4, or Sun386i System Over the Network

Sun-2, Sun-4 and Sun386i systems boot over the network in a similar fashion. However, the filename requested from a server must have a suffix that reflects the system architecture of the machine being booted. For these systems, the requested filename has the form:

ip-address.arch

where *ip-address* is the machine's Internet Protocol (IP) address in hex, and *arch* is a suffix representing its architecture. (Only Sun-3 systems may omit the *arch* suffix.) These filenames are restricted to 14 characters for compatibility with System V and other operating systems. Therefore, the architecture suffix is limited to 5 characters; it must be in upper case. At present, the following suffixes are recognized: **SUN2** for Sun-2 system, **SUN3** for Sun-3 system, **SUN4** for Sun-4 system, **S386** for Sun386i system, and **PCNFS** for PC-NFS. That file is typically a symbolic link to the client's boot program, normally **boot.sun2** in the same directory for a Sun-2 system, **boot.sun3** in the same directory for a Sun-3 system, or **boot.sun4** in the same directory for a Sun-4 system.

Note: a Sun-2 system boots from its server using one extra step. It broadcasts an ND request which is intercepted by the user-level **ndbootd (8C)** (see **ndbootd**(8C) server. This server sends back a standalone program that carries out the same TFTP request sequence as is done for all the other systems.

System Startup

Once the system is loaded and running, the kernel performs some internal housekeeping, configures its device drivers, and allocates its internal tables and buffers. The kernel then starts process number 1 to run **init**(8), which performs file system housekeeping, starts system daemons, initializes the system console, and begins multiuser operation. Some of these activities are omitted when **init** is invoked with certain *boot-flags*. These are typically entered as arguments to the boot command, and passed along by the kernel to **init**.

OPTIONS

device One of:

ie	Intel Ethernet
ec	3Com Ethernet
le	Lance Ethernet
sd	SCSI disk
st	SCSI 1/4" tape
mt	Tape Master 9-track 1/2" tape
xt	Xylogics 1/2" tape
xy	Xylogics 440/450/451 disk

c Controller number, **0** if there is only one controller for the indicated type of device.

u Unit number, **0** if only there is only one driver.

filename Name of a standalone program in the selected partition, such as **stand/diag** or **vmunix**. Note: *filename* is relative to the root of the selected device and partition. It never begins with '/' (slash). If *filename* is not given, the boot program uses a default value (normally **vmunix**). This is stored in the **vmunix** variable in the **boot** executable file supplied by Sun, but can be patched to indicate another standalone program loaded using **adb**(1).

−a Prompt interactively for the device and name of the file to boot. For more information on how to boot from a specific device, refer to *Installing SunOS 4.1*.

−v Verbose. Print more detailed information to assist in diagnosing diskless booting problems.

boot-flags The boot program passes all *boot-flags* to the kernel or standalone program. They are typically arguments to that program or, as with those listed below, arguments to programs that it invokes.

−b Pass the **−b** flag through the kernel to **init**(8) so as to skip execution of the **/etc/rc.boot** script.

−h Halt after loading the system.

−s Pass the **−s** flag through the kernel to **init**(8) for single-user operation.

−i *initname*
Pass the **−i** *initname* to the kernel to tell it to run *initname* as the first program rather than the default **/sbin/init.**

FILES

/boot	standalone boot program
/tftpboot/*address*	symbolic link to the boot program for the client whose Internet address, in uppercase hexadecimal, is *address*
/tftpboot/boot.sun3	Sun-3 first stage boot program
/tftpboot/boot.sun4	Sun-4 first stage boot program
/usr/etc/in.tftpd	TFTP server
/usr/mdec/installboot	program to install boot blocks from a remote host
/vmunix	kernel file that is booted by default
/etc/bootparams	file defining root and swap paths for clients

SEE ALSO

adb(1), **tftp**(1C) **bootparamd**(8), **init**(8), **kadb**(8S), **monitor**(8S), **mount**(8), **ndbootd**(8C), **rc**(8), **reboot**(8), **tftpd**(8C)

Installing SunOS 4.1
System and Network Administration

BUGS

On Sun-2 systems, the PROM passes in the default name **vmunix**, overriding the the boot program's patchable default.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

bootparamd – boot parameter server

SYNOPSIS

/usr/etc/rpc.bootparamd [**–d**]

DESCRIPTION

bootparamd is a server process that provides information to diskless clients necessary for booting. It first consults the local **/etc/bootparams** file for a client entry. If the local bootparams file does not exist, **bootparamd** consults the corresponding Network Interface Service (NIS) map.

bootparamd can be invoked either by **inetd**(8C) or by the user.

OPTIONS

–d Display the debugging information.

FILES

/etc/bootparams

SEE ALSO

inetd(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

C2conv, C2unconv – convert system to or from C2 security

SYNOPSIS

C2conv

C2unconv

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

C2conv converts a standard SunOS system to operate with C2-level security.

The program prompts for information regarding the Secure NFS option, client systems (if the system is an NFS server for diskless clients), audit devices (if it is an audit file server), and names of file systems (if there is a remote audit server). The program also requests certain information for the **audit_control**(5) file; default values may be used for audit flags and for the "minfree" value. Finally, it requests the mail address to be used (by **mail**(1)) when C2 administrative tasks are required. The default address is **root** for the host being converted.

Once it has this information, **C2conv** uses it to set up the necessary files for a C2 secure system, reporting on its progress as it proceeds.

C2unconv backs out the changes made to **/etc/passwd** and **/etc/group**. It does not back out changes to other files.

FILES

/etc/passwd
/etc/group
/etc/fstab

SEE ALSO

audit_control(5)

NAME

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**–v** . . .] [**–V**] [**–1**] [**–w** *width*] *filename* . . .

SYNOPSIS

/usr/5bin/captoinfo [**–v** . . .] [**–V**] [**–1**] [**–w** *width*] *filename* . . .

AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

captoinfo converts the **termcap**(5) terminal description entries given in *filename* into **terminfo**(5V) source entries, and writes them to the standard output along with any comments found in that file. A description that is expressed as relative to another description (as specified in the **termcap** *tc*= capability) is reduced to the minimum superset before being written.

If no *filename* is given, then the environment variable **TERMCAP** is used for the filename or entry. If **TERMCAP** is a full pathname to a file, only the terminal-name is specified in the environment variable **TERM** is extracted from that file. If that environment variable is not set, then the file **/etc/termcap** is read.

OPTIONS

–v Verbose. Print tracing information on the standard error as the program runs. Additional **–v** options increase the level of detail.

–V Version. Display the version of the program on the standard error and exit.

–1 Print fields one-per-line. Otherwise, fields are printed several to a line, to a maximum width of 60 characters.

–w *width*
Change the output to *width* characters.

FILES

/usr/share/lib/terminfo/?/* compiled terminal description database
/etc/termcap

SEE ALSO

curses(3V), **termcap**(5), **terminfo**(5V), **infocmp**(8V), **tic**(8V)

DIAGNOSTICS

tgetent failed with return code n
The termcap entry is not valid. In particular, check for an invalid '**tc=**' entry.

unknown type given for the termcap code *cc*.
The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code *cc*.
The boolean **termcap** entry *cc* was entered as a numeric or string capability.

the boolean (numeric, string) termcap code *cc* **is not a valid name.**
An unknown **termcap** code was specified.

tgetent failed on TERM=term.
The terminal type specified could not be found in the **termcap** file.

TERM=term: cap *cc* **(info** *ii***) is**
The **termcap** code was specified as a null string. The correct way to cancel an entry is with an '**@**', as in '**:bs@:**'. Giving a null string could cause incorrect assumptions to be made by the software which uses **termcap** or **terminfo**.

a function key for *cc* was specified, but it already has the value
: *vv*. When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name *cc* was specified in the ko termcap capability.
: A key was specified in the **ko** capability which could not be handled.

the *vi* character v (info *ii*) has the value *xx*, but ma gives n.
: The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

the unknown *vi* key v was specified in the ma termcap capability.
: A **vi**(1) key unknown to **captoinfo** was specified in the **ma** capability.

Warning: termcap sg (*nn*) and termcap ug (*nn*) had different values.
: **terminfo** assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.
: The parameterized string being created should be rewritten by hand.

Null termname given.
: The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

cannot open *filename* for reading.
: The specified file could not be opened.

WARNINGS

Certain **termcap** defaults are assumed to be true. The bell character (**terminfo bel**) is assumed to be ^G. The linefeed capability (**termcap nl**) is assumed to be the same for both **cursor_down** and **scroll_forward** (**terminfo cud1** and **ind**, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for **termcap** fields such as **cursor_position** (**termcap cm**, **terminfo cup**) can sometimes produce a string that may not be optimal. In particular, the rarely used **termcap** operation *%n* produces strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a **termcap** entry, a hold-over from an earlier version of the system, has been removed.

NAME

catman – create the cat files for the manual

SYNOPSIS

/usr/etc/catman [**–nptw**] [**–M directory**] [**–T** *tmac.an*] [*sections*]

DESCRIPTION

catman creates the preformatted versions of the on-line manual from the **nroff(1)** input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, **catman** recreates the **whatis** database.

If there is one parameter not starting with a '–', it is taken to be a list of manual sections to look in. For example

catman 123

only updates manual sections **1**, **2**, and **3**.

If an unformatted source file contains only a line of the form '**.so manx/yyy.x**', a symbolic link is made in the catx or fmtx directory to the appropriate preformatted manual page. This feature allows easy distribution of the preformatted manual pages among a group of associated machines with **rdist**(1), since it makes the directories of preformatted manual pages self-contained and independent of the unformatted entries.

OPTIONS

–n Do not (re)create the **whatis** database.

–p Print what would be done instead of doing it.

–t Create **troff**ed entries in the appropriate **fmt** subdirectories instead of **nroff**ing into the **cat** subdirectories.

–w Only create the **whatis** database that is used by **whatis**(1) and the **man**(1) **–f** and **–k** options. No manual reformatting is done.

–M Update manual pages located in the specified **directory** (**/usr/man** by default).

–T Use **tmac.an** in place of the standard manual page macros.

ENVIRONMENT

TROFF The name of the formatter to use when the **–t** flag is given. If not set, '**troff**' is used.

FILES

/usr/[share]/man	default manual directory location
/usr/[share]/man/man?/*.*	raw (nroff input) manual sections
/usr/[share]/man/cat?/*.*	preformatted **nroff**ed manual pages
/usr/[share]/man/fmt?/*.*	preformatted **troff**ed manual pages
/usr/[share]/man/whatis	**whatis** database location
/usr/lib/makewhatis	command script to make whatis database

SEE ALSO

apropos(1), **man**(1), **nroff**(1), **rdist**(1), **troff**(1), **whatis**(1)

NOTES

If the **–n** option is specified, the **/usr/man/whatis** database is not created and the **apropos**, **whatis**, '**man –f**', and '**man –k**' commands will fail.

DIAGNOSTICS

man?/xxx.? (.so'ed from man?/yyy.?): No such file or directory
The file outside the parentheses is missing, and is referred to by the file inside them.

target of .so in man?/xxx.? must be relative to /usr/man
catman only allows references to filenames that are relative to the directory **/usr/man**.

opendir:man?: No such file or directory
A harmless warning message indicating that one of the directories **catman** normally looks for is missing.

***.*: No such file or directory**
A harmless warning message indicating **catman** came across an empty directory.

NAME

change_login – control screen blanking and choice of login utility

SYNOPSIS

change_login

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

To prolong the life of your monitor, your Sun386i system turns off the screen display if you have not used the keyboard or mouse for 30 minutes or more. To see the screen again, simply move the mouse on the pad or press any key. This feature is normally enabled automatically when you log in, but you can control it using the **change_login** command as explained below.

This command also determines whether you log into your workstation using the Sun386i login screen, **logintool**(8) or through a traditional **login:** prompt.

The screen blanking choices available with **change_login** are:

1. Logintool and Sun Logo screenblank
Enables screen blanking. When blank, the system displays the Sun logo moving randomly around an otherwise dark screen.

2. Logintool and video-off screenblank
Shuts off the video output to your monitor when the screen goes blank. This is the most efficient type of screen blanking. The Desktop is almost instantly redisplayed when you move the mouse or begin typing.

3. Logintool and no screenblank
Retains the login screen, but disables screen blanking.

4. No Logintool and no screenblank
Disables both the login screen and screen blanking.

EXAMPLE

The following is an example of **change_login**. Notice you must be super-user to use this command.

```
example# change_login

This program will check what login and screenblank
options are set on this workstation, and allow you
to choose other options, if you are logged in as superuser.

Do you want to do this? [y or n]: y

This workstation is set up to use logintool and
a screenblank program that displays a Sun logo graphic.

These are the available options:

+ 1. Logintool and Sun Logo screenblank
  2. Logintool and video-off screenblank
  3. Logintool and no screenblank
  4. No Logintool and no screenblank

+ indicates the current configuration

You must be logged in as superuser to change the current setting.
```

Follow the instructions in *Sun386i System Setup and Maintenance* or *Sun386i Advanced Administration* to shut down and then restart your system. The setting chosen in the above example will not be enabled until you have restarted your system.

SEE ALSO

login(1), **screenblank**(1), **su**(1V), **logintool**(8)

Sun386i Advanced Skills
Sun386i System Setup and Maintenance
Sun386i Advanced Administration

NAME

chown – change owner

SYNOPSIS

/usr/etc/chown [**–fHR**] *owner*[*.group*] *filename* ...

DESCRIPTION

chown changes the owner of the *filenames* to *owner*. The owner may be either a decimal user ID (UID) or a login name found in the password file. An optional *group* may also be specified. The group may be either a decimal group ID (GID) or a group name found in the GID file.

Only the super-user can change owner, in order to simplify accounting procedures.

OPTIONS

–f Do not report errors.

–R Recursively descend into directories setting the ownership of all files in each directory encountered. When symbolic links are encountered, their ownership is changed, but they are not traversed.

FILES

/etc/passwd password file

SEE ALSO

chgrp(1), **chown**(2V), **group**(5), **passwd**(5)

NAME

chroot – change root directory for a command

SYNOPSIS

/usr/etc/chroot *newroot command*

DESCRIPTION

The given command is executed *relative* to the new root. The meaning of any initial '/' (slashes) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Input and output redirections on the command line are made with respect to the *original* root:

chroot newroot command >x

creates the file *x* relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a **chroot** is already in effect; the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2V)

BUGS

One should exercise extreme caution when referring to special files in the new root file system.

NAME

chrtbl – generate character classification table

SYNOPSIS

/usr/etc/chrtbl [*filename*]

DESCRIPTION

chrtbl converts a source description of a character classification table into a form that can be used by the character classification functions and multibyte functions (see **ctype**(3V) and **mblen**(3)). The source description is found in *filename*. If *filename* is not given, or just given as '–', **chrtbl** reads its source description from the standard input.

chrtbl creates one or two output files, the second file is only created if the **model** token is specified. By default, these files are created in the current working directory. The first file, named by the **chrclass** token, is always produced and contains the character classification information for all single-byte (7-bit and 8-bit) character code-sets described by one setting of the **LC_CTYPE** category of locale. The second file, created if the **model** token is specified, contains information relating to details of width and structure of the coded character set currently under definition. The second file is named by appending '**.ci**'. to the value specified by the **chrclass** token.

The first output file contains a binary form of the character classification information described in *filename*. It is structured in such a way that it can be used at run-time to replace the active version of the **ctype**[ ] array in the C-library, For it to be understood at run-time, the output file must be moved to the **/usr/share/lib/locale/LC_TYPE** or **/etc/locale** directory (see FILES below) by the super-user or a member of group **bin**. This file must be readable by user, group, and other; no other permission should be set.

filename contains a sequence of tokens in any order after the **chrclass** token, each separated by one or more NEWLINE characters or comment lines. The tokens recognized by **chrtbl** are as follows:

chrclass *name*
: *name* is the filename or pathname of the character classification file. This is a mandatory token. It must be the first token to be defined, and is usually given the name that relates to a valid setting of the **LC_CTYPE** category of locale.

model *name,args*
: This optional token chooses the type of character code-set announcement mechanism associated with the character classification table generated by **chrtbl**. The name of the file created by this token is the name specified by the **chrclass** token, concatenated with a '**.ci**'. The arguments to **model** must be one of the following:

euc x,y,z
: The model file contains information describing the required setting for the Extended Unix code-set announcement mechanism. *x,y,z* relate to the storage widths (in bytes) of EUC code-sets 1, 2 and 3 respectively.

xccs
: The model file contains information describing the Xerox Character Code Standard (XC1-3-3-0) announcement mechanism. There are no additional arguments required.

iso2022 g0,g1,g2,g3 x
: The model file contains information describing a generative version of the ISO-2022 code set announcement mechanism. The multibyte functions driven by this model are capable of handling the standard one or more byte escape sequences as well as all of the standard shift functions. The four arguments *g0,g1,g2,g3* define the default width (in bytes) of the four designations (respectively) available under ISO-2022, Maximum integer value of any of these arguments is 2. The fianl argument *x* is mandatory and must be set to either 7 or 8. It selects the default bit-width of each byte on input and output to/from the multibyte functions.

If the **model** token is declared without arguments, then it is assumed that there is a set of user-defined rules for character code-set announcement. This is noted in the output file and will be later used to fold in user-defined code into the multibyte functions in the C-library (see **mblen**(3)).

isupper Character codes to be classified as upper-case letters.

islower Character codes to be classified as lower-case letters.

isdigit Character codes to be classified as numeric.

isspace Character codes to be classified as a spacing (delimiter) character.

ispunct Character codes to be classified as a punctuation character.

iscntrl Character codes to be classified as a control character.

isblank Character code for the space character.

isxdigit Character codes to be classified as hexadecimal digits.

ul Relationship between upper- and lower-case characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as **0x61** in hexadecimal or **0141** in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The dash (–) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a RETURN is permitted after the backslash character.

The relationship between upper- and lower-case letters (**ul**) is expressed as ordered pairs of octal and hexadecimal constants:

<upper-case_character lower-case_character>

These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLES

The following is an example of an input file used to create the ASCII code set definition table on a file named **ascii**.

```
chrclass    ascii
isupper     0x41 – 0x5a
islower     0x61 – 0x7a
isdigit     0x30 – 0x39
isspace     0x20 0x9 – 0xd
ispunct     0x21 – 0x2f 0x3a – 0x40 \
            0x5b – 0x60 0x7b – 0x7e
iscntrl     0x0 – 0x1f 0x7f
isblank     0x20
isxdigit    0x30 – 0x39 0x61 – 0x66 \
            0x41 – 0x46
ul          <0x41 0x61> <0x42 0x62> <0x43 0x63> \
            <0x44 0x64> <0x45 0x65> <0x46 0x66> \
            <0x47 0x67> <0x48 0x68> <0x49 0x69> \
            <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c> \
            <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f> \
            <0x50 0x70> <0x51 0x71> <0x52 0x72> \
```

**<0x53 0x73> <0x54 0x74> <0x55 0x75> **
**<0x56 0x76> <0x57 0x77> <0x58 0x78> **
<0x59 0x79> <0x5a 0x7a>

FILES

/usr/share/lib/locale/LC_CTYPE/* run-time location of the character classification tables generated by **chrtbl**

/etc/locale/LC_CTYPE/* location for private versions of the classification tables generated by **chrtbl**

SEE ALSO

ctype(3V), **environ**(5V)

DIAGNOSTICS

The error messages produced by **chrtbl** are intended to be self-explanatory. They indicate input errors in the command line or syntactic errors encountered within the input file.

NAME

client – add or remove diskless Sun386i systems

SYNOPSIS

client [**–a** *arch*] [**–h** *hostid*] [**–o** *os*] [**–q**] [**–t** *minutes*] **add** *bootserver client etheraddress ipaddress*

client remove *client*

client modify *client* [**diskful | diskless | slave**]

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

client can be used to manually add and remove diskless clients of a PNP boot server. After successful completion of the command, the diskless client can boot. Only users in the *networks* group (group 12) on the boot server are allowed to change configurations using this utility. **client** can be invoked from any system on the network.

The boot server of a system is the only machine truly required for that system to boot to the point of allowing user logins; it must accordingly provide name, booting, and time services. Diskless clients can provide none of these services themselves. Diskful clients, however can provide most of their own boot services. Network clients only need name and time services from the network, and can use any boot server.

To add a diskless client, use the **add** operation. To remove a diskless, diskful, or network client, use the **remove** operation. To change a system's network role, use the **modify** operation.

A server can reject a configuration request if it is disallowed by the contents of the **bootservers** map (e.g., too many clients would be configured, or too little free space would be left on the server), or if no system software for the client is available.

OPTIONS

–a *arch*
Specifies the architecture code of the client; it defaults to **s386**. (Note: architecture codes are different from architecture names. Architecture codes are used in diskless booting, and are at most five characters in length, while architecture names can be longer.)

–h *hostid*
Specifies the host ID of the client; if supplied, it is used as the root password for the system. It defaults to the null string.

–o *os*
Specifies the operating system; defaults to 'unix'. This is currently used only to construct the system's *publickey* data, where applicable; this is never done if the system has no *hostid* specified.

–q
Quiet. Displays only error messages.

–t *minutes*
Sets the RPC timeout to the number of minutes indicated; this defaults to 15 minutes. If the bootserver takes more time than this to complete, **client** will exit. Unless the server has already completed setup, but not yet sent status to **client**, this will cause the bootserver to back out of the setup, deallocating all assigned resources.

SEE ALSO

publickey(5) **netconfig**(8C), **pnpd**(8C)

BUGS

Unless the *hostid* is assigned, the root filesystem for the diskless client is not set up beyond copying the **proto** and **boot** files into it. This means that **netconfig** will often handle other parts of the setup.

NAME

clri – clear inode

SYNOPSIS

/usr/etc/clri *filesystem i-number*...

DESCRIPTION

Note: **clri** has been superseded for normal file system repair work by **fsck**(8).

clri writes zeros on the inodes with the decimal *i-numbers* on the *filesystem*. After **clri**, any blocks in the affected file will show up as "missing" in an **icheck**(8) of the *filesystem*.

Read and write permission is required on the specified file system device. The inode becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(8) **fsck**(8)

BUGS

If the file is open, **clri** is likely to be ineffective.

NAME

colldef – convert collation sequence source definition

SYNOPSIS

/usr/etc/colldef *filename*

DESCRIPTION

colldef converts a collation sequence source definition into a format usable by the **strxfrm()** and **strcoll**(3) functions. It is used to define the many ways in which strings can be ordered and collated.

colldef reads the collation sequence source definition from the standard input and stores the converted definition in *filename*.

The collation sequence definition specifies a set of collating elements and the rules defining how strings containing these should be ordered. This is most useful for different language definitions. The rules provide the following capabilities:

1-to-Many mapping
: A single character is mapped into a string of collating elements.

Many-to-1 mapping
: A string of two or more characters is mapped as a single collating element.

Null string mapping
: A character, or string of characters, is mapped to a null collating element (that is, will be ignored).

Equivalence class definition.
: A collection of characters that have the same value.

Secondary ordering within equivalence class.

USAGE

The following keywords may be used in the input file *filename*.

charmap
: Optional keyword. Defines where a mapping of the character and collating element symbols to the actual character encoding can be found.

substitute
: Optional keyword. Defines a one-to-many mapping between a single byte and a character string.

order
: Mandatory keyword. Defines the primary and secondary ordering of collating elements within this collation table.

EXIT STATUS

colldef exits with the following values:

0
: No errors were found and the output was successfully created.

>0
: Errors were found.

FILES

/etc/locale/LC_COLLATE/*locale/domain*
: standard private location for collation orders under the *locale* locale

/usr/share/lib/locale/LC_COLLATE
: standard shared location for collation orders under the *locale* locale

SEE ALSO

strcoll(3)
System Services Overview

NAME

comsat, in.comsat – biff server

SYNOPSIS

/usr/etc/in.comsat

DESCRIPTION

comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by **inetd**(8C), and times out if inactive for a few minutes.

comsat listens on a datagram port associated with the **biff**(1) service specification (see **services**(5)) for one line messages of the form

> **user@mailbox-offset**

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a '**biff y**'), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the **From**, **To**, **Date**, or **Subject** lines are not printed when displaying the message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1), **services**(5), **inetd**(8C)

BUGS

The message header filtering is prone to error.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config – build system configuration files

SYNOPSIS

/usr/etc/config [**–fgnp**] [**–o** *obj_dir*] *config_file*

DESCRIPTION

config does the preparation necessary for building a new system kernel with **make**(1). The *config_file* named on the command line describes the kernel to be made in terms of options you want in your system, size of tables, and device drivers to be included. When you run **config**, it uses several input files located in the current directory (typically the **conf** subdirectory of the system source including your *config_file*). The format of this file is described below.

If the directory named *../config_file* does not exist, **config** will create one. One of **config**'s output files is a makefile which you use with **make**(1) to build your system.

You use **config** as follows. Run **config** from the **conf** subdirectory of the system source (in a typical Sun environment, from **/usr/share/sys/sun[2 3 4]/conf**):

```
example# /usr/etc/config config_file
Doing a "make depend"
example# cd ../config_file
example# make
...lots of output...
```

While **config** is running watch for any errors. Never use a kernel which **config** has complained about; the results are unpredictable. If **config** completes successfully, you can change directory to the *../config_file* directory, where it has placed the new makefile, and use **make** to build a kernel. The output files placed in this directory include **ioconf.c**, which contains a description of I/O devices attached to the system; **mbglue.s**, which contains short assembly language routines used for vectored interrupts, a makefile, which is used by **make** to build the system; a set of header files (*device_name*.**h**) which contain the number of various devices that may be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for the root file system, swapping, and system dumps.

Now you can install your new kernel and try it out.

OPTIONS

–f Set up the makefile for fast builds. This is done by building a **vmunix.o** file which includes all the **.o** files which have no source. This reduces the number of files which have to be **stat**ed during a system build. This is done by prelinking all the files for which no source exists into another file which is then linked in place of all these files when the kernel is made. This makefile is faster because it does not **stat** the object files during the build.

–g Get the current version of a missing source file from its SCCS history, if possible.

–n Do not do the '**make depend**'. Normally **config** will do the '**make depend**' automatically. If this option is used **config** will print '**Don't forget to do a "make depend"**' before completing as a reminder.

–p Configure the system for profiling (see **kgmon**(8) and **gprof**(1)).

–o *obj_dir*
Use *../obj_dir* instead of **../OBJ** as the directory to find the object files when the corresponding source file is not present in order to generate the files necessary to compile and link your kernel.

USAGE

Input Grammar

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal numbers are specified to **config** in the same way they are specified to the C compiler, a number starting with **0x** is a hex number and a number starting with just a **0** is an octal number.

Comments are begin with a # character, and end at the next NEWLINE. Lines beginning with TAB characters are considered continuations of the previous line. Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

machine "*type*"
: This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type*s for a Sun system are **sun2, sun3, sun4,** and **sun386.** Note: the double quotes around *type* are part of the syntax, and must be included.

cpu "*type*"
: This system is to run on the cpu type specified. More than one cpu type can appear in the config file. Legal *type*s for a **sun2** machine are noted in the annotated config file in *Installing SunOS 4.1.*

ident *name*
: Give the system identifier — a name for the machine or machines that run this kernel. Note that *name* must be enclosed in double quotes if it contains both letters and digits. Also, note that if *name* is **GENERIC**, you need not include the '**options GENERIC**' clause in order to specify '**swap generic**'.

maxusers *number*
: The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

options *optlist*
: Compile the listed options into the system. Options in this list are separated by commas. A line of the form:

    options FUNNY , HAHA

  yields

    –DFUNNY –DHAHA

  to the C compiler. An option may be given a value, by following its name with = (equal sign) then the value enclosed in (double) quotes. None of the standard options use such a value.

  In addition, options can be used to bring in additional files if the option is listed in the **files** files. All options should be listed in upper case. In this case, no corresponding *option*.**h** will be created as it would be using the corresponding *pseudo-device* method.

config *sysname config_clauses . . .*
: Generate a system with name *sysname* and configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, and a disk partition to which system dumps should be made. All but the root device specification may be omitted; **config** will assign default values as described below.

  root A root device specification is of the form '**root on** *xy0d*'. If a specific partition is omitted — for example, if only **root on xy0** is specified — the '**a**' partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

  swap To specify a swap partition, use a clause of the form: '**swap on** *partition*'. Swapping areas may be almost any size. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, '**swap on** *xy0b* **size 99999**' would configure a swap partition with 99999 sectors. If **swap generic** or no *partition* is specified with **on**, partition b on the root device is used. For dataless clients, use '**swap on type nfs**'.

To configure multiple swap partitions, specify multiple '**swap on**' clauses. For example:

config vmunix swap on xy0 swap on xy1

dumps The location to which system dumps are sent may be specified with a clause of the form '**dumps on** *xy1*'. If no dump device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the '**b**' partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time. Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by **savecore**(8).

Device names specified in configuration clauses are mapped to block device major numbers with the file **devices.***machine*, where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form '**major** *x* **minor** *y*'.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, a Xylogics 450 Disk Controller at address 0xee40 in the Multibus I/O space). These lines have the following format:

dev_type dev_name **at** *con_dev more_info*.

dev_type is either **controller, disk, tape, device,** or **pseudo-device.** These types have the following meanings:

controller A disk or tape controller.

disk or **tape** Devices connected to a controller.

device Something "attached" to the main system bus, like a cartridge tape interface.

pseudo-device A software subsystem or driver treated like a device driver, but without any associated hardware. Current examples are the pseudo-tty driver and various network subsystems. For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create.

dev_name is the standard device name and unit number (if the device is not a **pseudo-device**) of the device you are specifying. For example, **xyc0** is the *dev_name* for the first Xylogics controller in a system; **ar0** names the first quarter-inch tape controller.

con_dev is what the device you are specifying is connected to. It is either nexus?, a bus type, or a controller. There are several bus types which are used by **config** and the kernel.

The different possible bus types are:

obmem	On board memory
obio	On board io
mbmem	Multibus memory (**sun2** system only)
mbio	Multibus io (**sun2** system only)
vme16d16 (vme16)	16 bit VMEbus/ 16 bit data
vme24d16 (vme24)	24 bit VMEbus/ 16 bit data
vme32d16	32 bit VMEbus/ 16 bit data (**sun3** system only)
vme16d32	16 bit VMEbus/ 32 bit data (**sun3** system only)
vme24d32	24 bit VMEbus/ 32 bit data (**sun3** system only)
vme32d32 (vme32)	32 bit VMEbus/ 32 bit data (**sun3** system only)

All of these bus types are declared to be connected to nexus. The devices are hung off these buses. If the bus is wildcarded, then the autoconfiguration code will determine if it is appropriate to probe for the device on the machine that it is running on. If the bus is numbered, then the autoconfiguration code will only look for that device on machine type **N**. In general, the Multibus and VMEbus bus types are always wildcarded.

more_info is a sequence of the following:

csr *address* — Specify the address of the **csr** (command and status registers) for a device. The **csr** addresses specified for the device are the addresses within the bus type specified.

The **csr** address must be specified for all controllers, and for all devices connected to a main system bus.

drive *number* — For a disk or tape, specify which drive this is.

flags *number* — These flags are made available to the device driver, and are usually read at system initialization time.

priority *level* — For devices which interrupt, specify the interrupt level at which the device operates.

vector *intr number* [*intr number* . . .]
For devices which use vectored interrupts on VMEbus systems, *intr* specify the vectored interrupt routine and *number* the corresponding vector to be used (0x40-0xFF).

A **?** may be substituted for a number in two places and the system will figure out what to fill in for the **?** when it boots. You can put question marks on a *con_dev* (for example, at virtual '?'), or on a drive number (for example, drive '?'). This allows redundancy, as a single system can be built which will boot on different hardware configurations.

The easiest way to understand **config** files it to look at a working one and modify it to suit your system. Good examples are provided in *Installing SunOS 4.1*.

FILES

Files in **/usr/share/sys/sun[2 3 4]/conf** which may be useful for developing the *config_file* used by **config** are:

GENERIC — These are generic configuration files for either a Sun-2 or Sun-3 system. They contain all possible device descriptions lines for the particular architecture.

README — File describing how to make a new kernel.

As shipped from Sun, the files used by **/usr/etc/config** as input are in the **/usr/include/sys/conf** directory:

config_file	System-specific configuration file
Makefile.src	Generic prototype makefile for Sun–[23] systems
files	List of common files required to build a basic kernel
devices	Name to major device mapping file for Sun–[23] systems

/usr/etc/config places its output files in the **..**/*config_file* directory:

mbglue.s	Short assembly language routines used for vectored interrupts
ioconf.c	Describes I/O devices attached to the system
makefile	Used with **make**(1) to build the system
device_name.**h**	a set of header files (various *device_name*'s) containing devices which can be compiled into the system

SEE ALSO

gprof(1), **make**(1), **kgmon**(8), **savecore**(8)

The SYNOPSIS portion of each device entry in Section 4 of this manual.

Installing SunOS 4.1
System and Network Administration

NAME

copy_home – fetch default startup files for new home directories

SYNOPSIS

/home/*groupname***/copy_home /home/***groupname* **/home/***username*

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Whenever **snap**(1) is used to add a new user account, the **copy_home** script in the selected primary group's home directory is executed to copy the default files to the new user's home directory, and also perform any additional custom setup.

copy_home copies default environment files, such as **.cshrc**, **.login**, and **.orgrc**, from a group's **defaults** directory to a new user's home directory. It is started by **user_agentd**(8) when **snap**(1) is used to create new home directories on a Sun386i home directory server.

Every new group created by **snap**(1) has a home directory, which can be accessed using **/home/***groupname*. **user_agentd**(8) copies the contents of the Sun386i's default group, **/home/users**, into the home directory of the new group. This includes the **Welcome.txt** file, the **copy_home** script, and the **defaults** directory . **copy_home** can be modified to customize the default setup environment for new users in the group.

SEE ALSO

snap(1), **user_agentd**(8)

Sun386i SNAP Administration
Sun386i Advanced Administration

NAME

crash – examine system images

SYNOPSIS

/etc/crash [**–d** *dump-file*] [**–n** *namelist-file*] [**–w** *output-file*]

DESCRIPTION

crash examines the memory image of a live or a crashed system kernel. It displays the values of system control structures, tables, and other pertinent information.

OPTIONS

–d *dump-file* Specify the file containing the system memory image. The default is **/dev/mem**.

–n *namelist-file*
: Specify the text file containing the symbol table for symbolic access to the memory image. The default is **/vmunix**. If a system image from another machine is to be examined, the image file must be copied from that machine.

–w *output-file* Specify a file for **crash** output. The default is the standard output.

USAGE

For commands that pertain to a process, the default process is the one currently running on a live system, or the one that was running at the time the system crashed.

If the contents of a table are being dumped, the default is all active table entries.

Numeric Notation

Depending on the command, numeric arguments are assumed to be in a specific base. Counts are assumed to be decimal. Addresses are always hexadecimal. Table addresses larger than the size of the specified table are interpreted as hexadecimal addresses; smaller arguments are assumed to be in decimal. The default base of any argument may be overridden; the C conventions for designating the base of a number are recognized. (A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by **0x** and as octal if it is preceded by **0**. Decimal override is designated by **0d**, and binary by **0b**.)

Expressions

Many commands accept several forms of an argument. Requests for table information accept a table entry number, a physical address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a–b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number. The operator may be "+" (plus sign), "–" (minus sign), "*" (multiplication symbol), "/" (division symbol), "**&**" (logical AND), or "|" (logical OR). An operand which is a number should be preceded by a radix prefix if it is not a decimal number (**0** for octal, **0x** for hexidecimal, **0b** for binary). The expression must be enclosed in '( )' (parentheses). Other commands accept any of these argument forms that are meaningful.

Two abbreviated arguments to **crash** commands are used throughout. Both accept data entered in several forms. A *table_entry* argument may be an address, symbol, range or expression that resolves to one of these. A *start_addr* argument may be an address, symbol, or expression that resolves to one of those.

Commands

? [**–w** *filename*]
: List available commands.

  –w *filename*
  : Redirect the output of a command to the named file. Corresponds to the **redirect** command.

!*command*
: Escape to the shell to execute a command.

adv [**−ep**] [**−w** *filename*] [*table_entry*] . . .
Print the advertise table.

−e Display every entry in a table.

−p Interpret all address arguments in the command line as physical addresses. With this option, all address and symbol arguments explicitly entered on the command line are interpreted as physical addresses. Corresponds to the **mode** command.

as [−wfilename] [**−p**] *proc_entry* | *#pid* [s]]
Print the address space table.

base [**−w** *filename*] *number* . . .
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that indicates its radix as follows: **0x**, hexidecimal; **0**, octal; and **0b**, binary.

buffer [**−w** *filename*] [*−format*] *bufferslot*
buffer [**−p**] [**−w** *filename*] [*−format*] *start_addr*
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **−b**, byte; **−c**, character; **−d**, decimal; **−x**, hexadecimal; **−o**, octal; **−r**, directory; and **−i**, inode. If no format is given, the previous format is used. The default format at the beginning of a **crash** session is hexadecimal.

bufhdr [**−fp**] [**−w** *filename*] [*table_entry*] . . .
Alias: **buf**.
Print system buffer headers.

−f Display the full structure.

callout [**−w** *filename*]
Alias: **c**.
Print the callout table.

ctx [**−w***filename*] [[**−p**] *tbl_entry* . . .]
Print the context table.

dbfree [**−w** *filename*]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**−ep**] [**−w** *filename*] [*dblk_addr*] . . .
Print allocated streams data block headers. If the class option (**−c**) is used, only data block headers for the class specified will be printed.

defproc [**−c**] [**−w** *filename*]
defproc [**−w** *filename*] [*slot*]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**−c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a **crash** session, the process slot is set to the current process.

ds [**−w** *filename*] *virtual_address* . . .
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [**−ep**] [**−w** *filename*] [*table_entry*] . . .
Alias: **f**.
Print the file table.

findaddr [**−w** *filename*] *table slot*
Print the address of *slot* in *table*. Only tables available to the **size** command are available to **findaddr**.

gdp [**−efp**] [**−w** *filename*] [*table_entry*] ...
Print the gift descriptor protocol table.

help [**−w** *filename*] *command* ...
Print a description of the named command, including syntax and aliases.

inode [**−f**] [**−w** *filename*] [*table_entry*] ...
Alias: **i**.
Print the inode table, including file system switch information.

kfp [**−r**] [**−s** *process*] [**−w** *filename*]
kfp [**−s** *process*] [**−w** *filename*] [*value*]
Print the frame pointer for the start of a kernel stack trace. The **kfp** value can be set using the value argument or the reset option (**−r**), which sets the **kfp** through the nvram. If no argument is entered, the current value of the **kfp** is printed.

−s *process* Specify a process slot other than the default. Corresponds to the **defproc** command.

linkblk [**−ep**] [**−w** *filename*] [*table_entry*] ...
Print the **linkblk** table.

map [**−w** *filename*] *mapname* ...
Alias: **m**.
Print the map structure of *mapname*.

mbfree [**−w** *filename*]
Print free streams message block headers.

mblock [**−ep**] [**−w** *filename*] [*mblk_addr*] ...
Print allocated streams message block headers.

mode [**−w** *filename*] [*mode*]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a **crash** session, the mode is virtual.

mount [**−p**] [**−w** *filename*] [*table_entry*] ...
Alias: **m**.
Print the mount table.

nm [**−w** *filename*] *symbol* ...
Print value and type for the given symbol.

od [**−p**] [**−w** *filename*] [*−format*] [*−mode*] [**−s** *process*] *start_addr* [*count*]
Alias: **rd**.
Print *count* values starting at the start address in one of the following formats:

−c	character
−d	decimal
−x	hexadecimal
−o	octal
−a	ASCII
−h	hexadecimal character

and one of the following modes:

−l	long
−t	short
−b	byte

The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **−h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a **crash** session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

page [**−e**] [**−w***filename*] [[**−p**] *tbl_entry*] ...
Alias: **p**.
Print the page structures.

pcb [**−w** *filename*] [*process*]
Print the process control block. If no arguments are given, the active **pcb** for the current process is printed. **−ep**

pment [**−p**] [**−w***filename*] *tbl_entry* ...
Print the page map entry table (not available on machines with a **sun3x** kernel architecture).

pmgrp [**−w***filename*] [[**−p**] *tbl_entry* ...]
Print the page map group table (not available on machines with a **sun3x** kernel architecture).

proc [**−fp**] [**−w** *filename*] [*#pid*] ... [*table_entry*] ...
proc [**−fr**] [**−w** *filename*]
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process IDs (PID) may be entered. Each PID must be preceded by a '#' (pound sign). Alternatively, process table information for runnable processes may be specified with the runnable option (**−r**).

qrun [**−w** *filename*]
Print the list of scheduled streams queues.

queue [**−p**] [**−w** *filename*] [*queue_addr*] ...
Print stream queues.

quit Alias: **q**.
Terminate the **crash** session.

rcvd [**−efp**] [**−w** *filename*] [*table_entry*] ...
Print the receive descriptor table.

redirect [**−c**] [**−w** *filename*]
redirect [**−w** *filename*] [*filename*]
Alias: **rd**.
Used with a name, redirects output of a **crash** session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**−c**) closes the previously set file and redirects output to the standard output. To pipe output from a single **crash** command, use an exclamation point followed by a shell command:
crash-command **!** *shell-command*

This is not available when −w is in effect.

search [**−p**] [**−m** *mask*] [−s *process*] [**−w** *filename*] *pattern start_addr length*
Alias: **s**.
Print the words in memory that match *pattern*, beginning at the start address for *length* words. The mask is ANDed (&) with each memory word and the result compared against the pattern. The mask defaults to **0xffffffff**.

seg [**−w***filename*] [[**−p**] *proc_entry*]
seg [**−w***filename*] [*#procid* ...]
Print the segment table of process.

segdata [−w*filename*] [[−**p**] *proc_entry*]
segdata [−w*filename*] [*#procid* ...]
Print the segment data of process.

size [−**x**] [−**w** *filename*] [*structure_name* ...]
Print the size of the designated structure. The −**x** option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [−**efp**] [−w *filename*] [*table_entry*] ...
Print the send descriptor table.

srmount [−**ep**] [−w *filename*] [*table_entry*] ...
Print the server mount table.

stack [−**u**] [−w *filename*] [*process*]
stack [−**k**] [−w *filename*] [*process*]
stack [−**p**] [−w *filename*] −**i** *start_addr*]
Alias: **s**.
Dump stack. The −**u** option prints the user stack. The −**k** option prints the kernel stack. The −**i** option prints the interrupt stack starting at the start address. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

status [−w *filename*]
Print system statistics.

stream [−**efp**] [−w *filename*] [*table_entry*] ...
Print the streams table.

strstat [−w *filename*]
Print streams statistics.

trace [−**r**] [−w *filename*] [*process*]
trace [−**p**] [−w *filename*] −**i** *start_addr*]
Alias: **t**.
Print stack trace. The **kfp** value is used with the −**r** option. The interrupt option prints a trace of the interrupt stack beginning at the start address. The interrupt stack trace and the stack trace for the current process are not available on a running system.

ts [−w *filename*] *virtual_address* ...
Print closest text symbol to the designated address.

user [−**f**] [−w *filename*] [*process*]
Alias: **u**.
Print the ublock for the designated process.

vfs [−w*filename*] [[−**p**] *tbl_entry* ...]
Print the **vfs** table.

vnode [−w*filename*] [[−**p**] *addr*]
Alias: **v**.
Print the vnode table.

vtop [−s *process*] [−w *filename*] *start_addr* ...
Print the physical address translation of the virtual start address.

FILES

/dev/mem system image of currently running system
/var/crash/*machine***/vmcore.***N*
/var/crash/*machine***/vmunix.***N*

SEE ALSO

savecore(8)

NAME

cron – clock daemon

SYNOPSIS

/usr/etc/cron

DESCRIPTION

cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in **crontab** files in the directory **/var/spool/cron/crontabs**. Users can submit their own **crontab** files using the **crontab**(1) command. Commands that are to be executed only once may be submitted using the **at**(1) command.

cron only examines **crontab** files and **at** command files during process initialization and when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since **cron** never exits, it should only be executed once. This is normally done by running **cron** from the initialization process through the file **/etc/rc**; see **init**(8). **/var/spool/cron/FIFO** is a FIFO file that **crontab** and **at** use to communicate with **cron**; it is also used as a lock file to prevent the execution of more than one **cron**.

FILES

/var/spool/cron	main cron directory
/var/spool/cron/FIFO	FIFO for sending messages to **cron**
/var/spool/cron/crontabs	directory containing **crontab** files

SEE ALSO

at(1), **crontab**(1), **sh**(1), **queuedefs**(5), **init**(8), **syslogd**(8)

DIAGNOSTICS

cron logs various errors to the system log daemon, **syslogd**(8), with a facility code of **cron**. The messages are listed here, grouped by severity level.

Err Severity

Can't create /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not create **/var/spool/cron/FIFO**.

Can't access /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not access **/var/spool/cron/FIFO**.

Can't open /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not open **/var/spool/cron/FIFO**.

Can't start cron - another cron may be running (/var/spool/cron/FIFO exists)
: **cron** found that /var/spool/cron/FIFO already existed when it was started; this normally means that **cron** had already been started, but it may mean that an earlier **cron** terminated abnormally without removing /var/spool/cron/FIFO.

Can't stat /var/spool/cron/FIFO: *reason*
: **cron** could not get the status of **/var/spool/cron/FIFO**.

Can't change directory to *directory*:*reason*
: **cron** could not change to *directory*.

Can't read *directory*:*reason*
: **cron** could not read *directory*.

error reading message: *reason*
: An error occurred when **cron** tried to read a control message from **/var/spool/cron/FIFO**.

message received — bad format
A message was successfully read by **cron** from **/var/spool/cron/FIFO**, but the message was not of a form recognized by **cron**.

SIGTERM
received **cron** was told to terminate by having a SIGTERM signal sent to it.

cron could not unlink /var/spool/cron/FIFO: *reason*
cron was told to terminate, but it was unable to unlink **/var/spool/cron/FIFO** before it terminated.

********* CRON ABORTED **********
cron terminated, either due to an error or because it was told to.

Can't open queuedefs file *file*:*reason*
cron could not open a *queuedefs* file.

I/O error reading queuedefs file *file*:*reason*
An I/O error occurred while **cron** was reading a *queuedefs* file.

Using default queue definitions
An error occurred while trying to read a *queuedefs* file; the default queue definitions will be used.

Can't allocate *number* **bytes of space**
An internal error occurred in **cron** while trying to allocate memory.

Info Severity

queue **queue max run limit reached**
There were more jobs running or to be run in the queue *queue* than the maximum number specified. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

MAXRUN (25) procs reached
There were more than 25 jobs running or to be run by **cron**. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

***** cron started *****
cron started running.

> CMD: *pid queue command job*
A **cron** job was started, in queue *queue*, with process ID *pid*. *command* is the command to be run. For **at** or **batch** jobs, *job* is the job number.

> *user pid queue time job*
A **cron** job was started for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. For **at** or **batch** jobs, *job* is the job number.

< *user pid queue time job status*
A **cron** job completed for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. For **at** or **batch** jobs, *job* is the job number. If the command terminated with a non-zero exit status or a signal, *status* indicates the exit status or signal.

Notice Severity

Can't fork
An attempt to **fork (2)** to run a new job failed; **cron** will attempt again after a 30-second delay.

Warning Severity

Can't stat queuedefs file *file*:*reason*
cron could not get the status of a *queuedefs* file in order to determine whether it has changed. **cron** will assume it has changed and will reread it.

NAME

dbconfig – initializes the dial box

SYOPNSIS

/usr/etc/dbconfig *serial-device*

DESCRIPTION

dbconfig opens the designated serial port and sets its baud, parity and transmission rates. It also removes all STREAMS modules already pushed upon it (such as **ttcompat**(4M) and **ldterm**(4M)) and pushes the dial box STREAMS module "db" onto the device. **db** then holds the stream open to maintain this configuration.

If the device **/dev/dialbox** has not been created and linked to the serial port, **dbconfig** will fail.

FILES

/dev/dialbox

SEE ALSO

db(4M), **ldterm**(4M), **ttcompat**(4M), **dialtest**(6)

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [**–i** *numbers*] [*filesystem*]

DESCRIPTION

Note: **dcheck** has been superseded for normal consistency checking by **fsck**(8).

dcheck reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, **dcheck** checks a set of default file systems.

dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fs(5), **fsck**(8), **clri**(8), **icheck**(8), **ncheck**(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

NAME

devinfo – print out system device information

SYNOPSIS

/usr/etc/devinfo [–v]

AVAILABILITY

This program is available on SPARCstation 1 systems only.

DESCRIPTION

devinfo displays the devices that the system knows about. The output will state the name of the device, its unit number, and whether a system device driver has claimed it. Since the internal system representation of this information is an *n*–ary tree, indentation is used to denote a parent-child relationship, and devices reported at the same indentation level are considered sibling devices.

OPTIONS

–v Report hardware specifications such as register addresses and interrupt priorities for each device.

EXAMPLE

The following example displays the format of **devinfo** output:

```
example% devinfo
Node 'Sun 4/60', unit #0 (no driver)
        Node 'options', unit #0 (no driver)
        Node 'zs', unit #0
        Node 'zs', unit #1
        Node 'fd', unit #0
        Node 'audio', unit #0
        Node 'sbus', unit #0
                Node 'dma', unit #0
                Node 'esp', unit #0
                        Node 'st', unit #1 (no driver)
                        Node 'st', unit #0
                        Node 'sd', unit #3
                        Node 'sd', unit #2
                        Node 'sd', unit #1
                        Node 'sd', unit #0
                Node 'le', unit #0
                Node 'bwtwo', unit #0
        Node 'auxiliary-io', unit #0
        Node 'interrupt-enable', unit #0
        Node 'memory-error', unit #0
        Node 'counter-timer', unit #0
        Node 'eeprom', unit #0
```

FILES

/dev/kmem to get kernel device information

NAME

devnm – device name

SYNOPSIS

/usr/etc/devnm [*name*] . . .

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

devnm identifies the special file associated with the mounted file system where each *name* argument resides. This command can be used to construct a mount table entry for the **root** file system.

EXAMPLE

If **/usr** is mounted on **/dev/dsk/c1d0s2**, then the command:

/usr/etc/devnm /usr

produces:

/dev/dsk/c1d0s2 usr

FILES

/dev/dsk/*
/etc/mtab

SEE ALSO

fstab(5) **mount**(8)

NAME

diskusg – generate disk accounting data by user

SYNOPSIS

diskusg [**–sv**] [**–p** *filename*] [**–u** *filename*] [*filename* ...]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *filename*, or the standard input if *filename* is omitted. **diskusg** displays one line per user on the standard output in the following format:

uid login #blocks

uid is the numerical user ID of the user. *login* is the user's login name. *#blocks* is the total number of disk blocks allocated to the user.

diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *filename*s are the special filenames of these devices.

The output of **diskusg** is normally the input to **acctdisk** (see **acct**(8)) which generates total accounting records that can be merged with other accounting records. **diskusg** is normally run in **dodisk** (see **acctsh**(8)).

OPTIONS

–s The input data is already in **diskusg** output format; combine all lines for a single user into a single line.

–v Print a list to the standard error of all files that are not charged to any user.

–p *filename* Use *filename* as the name of the password file to generate login names. **/etc/passwd** is used by default.

–u *filename* Write records to *filename* of files that are not charged to any user. Records consist of the special file name, the i-node number, and the user ID.

EXAMPLES

The following example generates daily disk accounting information:

```
for i in /dev/xy0a /dev/xy0g /dev/xy1g; do
        diskusg $i > dtmp.`basename $i` &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd used for user ID to login name conversions

SEE ALSO

acct(5), **acct**(8), **acctsh**(8)

NAME

dkctl – control special disk operations

SYNOPSIS

/usr/etc/dkctl *disk command*

DESCRIPTION

dkctl is used to enable or disable special disk operations. In particular the enabling or disabling of verified writes (write check functionality) is controlled by this program.

The *disk* specification here is a disk name of the form */dev/rxxnp*, where *xx* is the controller device abbreviation (xy, sd, etc.), *n* is the disk number, and *p* is the partition to which the operation applies. The *partition* specification is simply the letter used to identify that partition in the standard UNIX system nomenclature.

SUPPORTED COMMANDS

wchk This function enables write checking for disks that support it for the named disk partition. This means that for partitions of disks with this feature enabled, all writes are *verified* to have been correctly written on the disk. This operation emphasizes data reliability over performance, although for each implementation, the fastest reasonable method will be used (i.e., implemented in hardware, if possible).

–wchk This disables write check functionality for the named disk partition.

BUGS

Use of the **dkctl** command requires super-user permissions.

There are many other features this program could control, and may in the future.

FILES

/dev/r*xxnp*

SEE ALSO

dkio(4S), **sd**(4S), **xy**(4S)

NAME

dkinfo – report information about a disk's geometry and partitioning

SYNOPSIS

/usr/etc/dkinfo *disk* [*partition*]

DESCRIPTION

dkinfo gives the total number of cylinders, heads, and sectors or tracks on the specified *disk*, and gives this information along with the starting cylinder for the specified *partition*. If no *partition* is specified on the command line, **dkinfo** reports on all partitions.

The *disk* specification here is a disk name of the form *xxn*, where *xx* is the controller device abbreviation (ip, xy, etc.) and *n* is the disk number. The *partition* specification is simply the letter used to identify that partition in the standard UNIX system nomenclature. For example, '**/usr/etc/dkinfo xy0**' reports on the first disk in a system controlled by a Xylogics controller; '**/usr/etc/dkinfo xy0g**' reports on the seventh partition of such a disk.

EXAMPLE

A request for information on my local disk, an 84 MByte disk controlled by a Xylogics 450 controller, might look like this:

```
#/usr/etc/dkinfo xy0
xy0: Xylogics 450 controller at addr ee40, unit # 0
586  cylinders 7 heads 32 sectors/track
a: 15884 sectors (70 cyls, 6 tracks, 12 sectors)
starting cylinder 0
b: 33440 sectors (149 cyls, 2 tracks)
starting cylinder 71
c: 131264 sectors (586 cyls)
starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 81760 sectors (365 cyls)
starting cylinder 221
h: No such device or address
#
```

FILES

/dev/r*xxnp*

SEE ALSO

dkio(4S), **format**(8S)

NAME

dmesg – collect system diagnostic messages to form error log

SYNOPSIS

/usr/etc/dmesg [–]

DESCRIPTION

Note: **dmesg** is obsoleted by **syslogd**(8) for maintenance of the system error log.

dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed or logged by the system when errors occur. If the '–' flag is given, then **dmesg** computes (incrementally) the new messages since the last time it was run and places these on the standard output.

FILES

/var/adm/msgbuf scratch file for memory of '–' option

SEE ALSO

syslogd(8)

NAME

dname – print RFS domain and network names

SYNOPSIS

dname [**–adn**] [**–D** *domain*] [**–N** *netspec*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

dname prints or defines a host's Remote File Sharing (RFS) domain name or the network used by RFS as transport provider.

When **dname** is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time RFS is started. See **rfstart**(8).

If **dname** is used with no options, it defaults to '**dname –d**'.

You cannot use the **–D** or **–N** options while RFS is running.

OPTIONS

–a Print both the domain name and network name.

–d Print the domain name.

–n Print the network name.

–D *domain*
Set the domain name for the host. *domain* must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (–), and underscores (_). This option is restricted to the super-user.

–N *netspec*
Set the network specification used for RFS. *netspec* is the network device name, relative to the **/dev** directory. For example, the TCP transport device, **/dev/tcp** uses **tcp**. This option is restricted to the super-user.

SEE ALSO

rfstart(8)

NOTES

This domain name is not related to the Network Interface Service (NIS) domain name. Note: NIS was formerly known as Sun Yellow Pages (YP). The functionality of thc two remains the same; only the name has changed.

NAME

dorfs – initialize, start and stop RFS automatically

SYNOPSIS

dorfs init *domain netspec* [*address*]
dorfs start [–v]
dorfs stop

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

dorfs sets up necessary environment to run Remote File Sharing (RFS). You can also use it to start or stop RFS automatically, after its environment is initialized. The environment only needs to be set up once and **/usr/nserve/rfmaster** must exist before the environment is initialized. Descriptions of **/usr/nserve/rfmaster** are in **rfmaster**(5). You must be the super-user to run this command.

USAGE

Subcommands

init *domain netspec* [*address*]

domain is the name of the RFS domain. *netspec* is the name of a device file in the **/dev** directory which represents the streams-based transport provider on which RFS will run. Currently, **tcp** is the only accepted value for this field. *address* is the optional **tcp** port number on which the listener will listen. If unspecified, it defaults to **0x1450**. This subcommand only needs to be run once to initialize the environment. You do not need to rerun **dorfs** with the **init** argument, unless you want to change *netspec*. **/usr/nserve/rfmaster** must exists before you run this command to initialize the environment. To reinitialize the environment, you need to remove **/usr/nserve/***domain*, **/usr/nserve/***netspec*, **/var/net/nls/***netspec***/***address* and **/var/net/nls/***netspec***/dbf** beforehand.

start [–v]

Start RFS automatically. It also automatically advertises resources that are stored in **/etc/rstab** and mounts RFS resources that are stored in **/etc/fstab**.

–v Verify clients on mounts (see '**rfstart –v**').

stop

Takes down RFS by forced unmounting of all advertised resources, umounting all remotely mounted resources, executing **rfstop**, and stopping **listener**.

FILES

/etc/advtab
/etc/rstab
/var/net/nls/tcp/addr
/var/net/nls/tcp/dbf
/usr/nserve/domain
/usr/nserve/netspec
/usr/nserve/rfmaster

SEE ALSO

rfmaster(5), **dname**(8), **fumount**(8), **mount**(8), **nlsadmin**(8), **rfstart**(8), **rfstop**(8)

NAME

dump, rdump – incremental file system dump

SYNOPSIS

/usr/etc/dump [*options* [*arguments*]] *filesystem*
/usr/etc/dump [*options* [*arguments*]] *filename* ...

/usr/etc/rdump [*options* [*arguments*]] *filesystem*
/usr/etc/rdump [*options* [*arguments*]] *filename* ...

DESCRIPTION

dump backs up all files in *filesystem*, or files changed after a certain date, or a specified set of files and directories, to magnetic tape, diskettes, or files. *options* is a string that specifies **dump** options, as shown below. Any *arguments* supplied for specific options are given as subsequent words on the command line, in the same order as that of the *options* listed.

If **dump** is called as **rdump,** the dump device defaults to **dumphost:/dev/rmt8**.

If no *options* are given, the default is **9u**.

dump is normally used to back up a complete filesystem. To restrict the dump to a specified set of files and directories on one filesystem, list their names on the command line. In this mode the dump level is set to **0** and the **u** option is ignored.

OPTIONS

0–9 The "dump level." All files in the *filesystem* that have been modified since the last **dump** at a lower dump level are copied to the volume. For instance, if you did a "level **2**" dump on Monday, followed by a "level **4**" dump on Tuesday, a subsequent "level **3**" dump on Wednesday would contain all files modified or added since the "level **2**" (Monday) backup. A "level **0**" dump copies the entire filesystem to the dump volume.

a *archive-file*
Create a dump table-of-contents archive in the specified file, *archive-file*. This file can be used by **restore**(8) to determine whether a file is present on a dump tape, and if so, on which volume it resides. For further information on the use of a dump archive file, see **restore**(8).

b *factor* Blocking factor. Specify the blocking factor for tape writes. The default is 20 blocks per write. Note: the blocking factor is specified in terms of 512 bytes blocks, for compatibility with **tar**(1). The default blocking factor for tapes of density 6250BPI and greater is 64. The default blocking factor for cartridge tapes (c option specified) is 126. The highest blocking factor available with most tape drives is 126.

c Cartridge. Use a cartridge instead of the standard half-inch reel. This sets the density to 1000BPI, the blocking factor to 126, and the length to 425 feet. This option also sets the "inter-record gap" to the appropriate length. When cartridge tapes are used, and this option is *not* specified, **dump** will slightly miscompute the size of the tape. If the **b**, **d**, **s** or **t** options are specified with this option, their values will override the defaults set by this option.

d *bpi* Tape density. The density of the tape, expressed in BPI, is taken from *bpi*. This is used to keep a running tab on the amount of tape used per reel. The default density is 1600 except for cartridge tape. Unless a higher density is specified explicitly, **dump** uses its default density — even if the tape drive is capable of higher-density operation (for instance, 6250BPI). Note: the density specified should correspond to the density of the tape device being used, or **dump** will not be able to handle end-of-tape properly. The **d** option is not compatible with the **D** option.

D Diskette. Specify diskette as the dump media.

f *dump-file*
Dump file. Use *dump-file* as the file to dump to, instead of **/dev/rmt8.** If *dump-file* is specified as '–', dump to the standard output. If the file name argument is of the form *machine:device*, dump to a remote machine. Since **dump** is normally run by *root*, the name of the local machine must

appear in the **.rhosts** file of the remote machine. If the file name argument is of the form *user@machine:device*, **dump** will attempt to execute as the specified user on the remote machine. The specified user must have a **.rhosts** file on the remote machine that allows root from the local machine. If **dump** is called as **rdump,** the dump device defaults to **dumphost:/dev/rmt8**. To direct the output to a desired remote machine, set up an alias for **dumphost** in the file **/etc/hosts.**

n Notify. When this option is specified, if **dump** requires attention, it sends a terminal message (similar to **wall**(1)) to all operators in the "operator" group.

s *size* Specify the *size* of the volume being dumped to. When the specified size is reached, **dump** waits for you to change the volume. **dump** interprets the specified size as the length in feet for tapes, and cartridges and as the number of 1024 byte blocks for diskettes. The following are defaults:

tape	2300 feet
cartridge	425 feet
diskette	1422 blocks (Corresponds to a 1.44 Mb diskette, with one cylinder reserved for bad block information.)

t *tracks* Specify the number of tracks for a cartridge tape. On all Sun-2 systems the default is 4 tracks, although some Sun-2 systems have 9 track drives. On all other machines the default is 9 tracks. The **t** option is not compatible with the **D** option.

u Update the dump record. Add an entry to the file **/etc/dumpdates,** for each filesystem successfully dumped that includes the filesystem name, date, and dump level. This file can be edited by the super-user.

v After writing each volume of the dump, the media is rewound and is verified against the filesystem being dumped. If any discrepancies are found, dump will respond as if a write error had occurred; the operator will be asked to mount new media, and dump will attempt to rewrite the volume. Note that *any* change to the filesystem, even the update of the access time on a file will cause the verification to fail. Thus, the verify option can only be used on a quiescent filesystem.

w List the filesystems that need backing up. This information is gleaned from the files **/etc/dumpdates** and **/etc/fstab**. When the **w** option is used, all other options are ignored. After reporting, **dump** exits immediately.

W Like **w**, but includes all filesystems that appear in **/etc/dumpdates,** along with information about their most recent dump dates and levels. Filesystems that need backing up are highlighted.

FILES

/dev/rmt8	default unit to dump to
dumphost:/dev/rmt8	default remote unit to dump to if called as **rdump**
/dev/rst*	Sun386i cartridge tape dump device
/dev/rfd0a	Sun386i 1.44 megabyte 3.5-inch high density diskette drive dump device
/dev/rfdl0a	Sun386i 720 kilobyte 3.5-inch low density diskette drive dump device
/dev/rfd0c	Sun386i 1.44 megabyte 3.5-inch high density diskette drive dump device
/dev/rfdl0c	Sun386i 720 kilobyte 3.5-inch low density diskette drive dump device
/etc/dumpdates	dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*
/etc/hosts	

SEE ALSO

bar(1), **fdformat**(1), **tar**(1), **wall**(1), **dump**(5), **fstab**(5), **restore**(8), **shutdown**(8)

DIAGNOSTICS

While running, **dump** emits many verbose messages.

Exit Codes

0 Normal exit.

1 Startup errors encountered.

3 Abort – no checkpoint attempted.

BUGS

Fewer than 32 read errors on the file system are ignored.

Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It is recommended that incremental dumps also be performed with the system running in single-user mode.

dump does not support multi-file multi-volume tapes.

NOTES

Operator Intervention

dump requires operator intervention on these conditions: end of volume, end of dump, volume write error, volume open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** option, **dump** interacts with the operator on **dump**'s control terminal at times when **dump** can no longer proceed, or if something is grossly wrong. All questions **dump** poses *must* be answered by typing **yes** or **no**, as appropriate.

Since backing up a disk can involve a lot of time and effort, **dump** checkpoints at the start of each volume. If writing that volume fails for some reason, **dump** will, with operator permission, restart itself from the checkpoint after a defective volume has been replaced.

dump reports periodically, and in verbose fashion. Each report includes estimates of the percentage of the dump completed and how long it will take to complete the dump. The estimated time is given as *hours*:*minutes*.

Suggested Dump Schedule

It is vital to perform full, "level **0**", dumps at regular intervals. When performing a full dump, bring the machine down to single-user mode using **shutdown**(8). While preparing for a full dump, it is a good idea to clean the tape drive and heads.

Incremental dumps allow for convenient backup and recovery on a more frequent basis of active files, with a minimum of media and time. However there are some tradeoffs. First, the interval between backups should be kept to a minimum (once a day at least). To guard against data loss as a result of a media failure (a rare, but possible occurrence), it is a good idea to capture active files on (at least) two sets of dump volumes. Another consideration is the desire to keep unnecessary duplication of files to a minimum to save both operator time and media storage. A third consideration is the ease with which a particular backed-up version of a file can be located and restored. The following four-week schedule offers a reasonable tradeoff between these goals.

	Sun	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*
Week 1:	**Full**	5	5	5	5	3
Week 2:		5	5	5	5	3
Week 3:		5	5	5	5	3
Week 4:		5	5	5	5	3

Although the Tuesday — Friday incrementals contain "extra copies" of files from Monday, this scheme assures that any file modified during the week can be recovered from the previous day's incremental dump.

Process Priority of dump

dump uses multiple processes to allow it to read from the disk and write to the media concurrently. Due to the way it synchronizes between these processes, any attempt to run dump with a **nice** (process priority) of '–5' or better will likely make **dump** run *slower* instead of faster.

NAME

dumpfs – dump file system information

SYNOPSIS

/usr/etc/dumpfs *device*

DESCRIPTION

dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), **fsck**(8), **newfs**(8), **tunefs**(8)

NAME

edquota – edit user quotas

SYNOPSIS

/usr/etc/edquota [**–p** *proto-user*] *usernames*...

/usr/etc/edquota –t

DESCRIPTION

edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disk quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, **edquota** reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is **vi**(1) unless the **EDITOR** environment variable specifies otherwise.

Only the super-user may edit quotas. (In order for quotas to be established on a file system, the root directory of the file system must contain a file, owned by root, called **quotas**. See **quotaon**(8) for details.)

OPTIONS

–p Duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

–t Edit the soft time limits for each file system. If the time limits are zero, the default time limits in **<ufs/quota.h>** are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems

SEE ALSO

quota(1), **vi**(1), **quotactl**(2), **quotacheck**(8), **quotaon**(8), **repquota**(8)

BUGS

The format of the temporary file is inscrutable.

NAME

eeprom – EEPROM display and load utility

SYNOPSIS

eeprom [–] [**–c**] [**–i**] [**–f** *device*] [*field*[=*value*] . . .]

SYNOPSIS — SPARCstation 1 SYSTEMS

eeprom [–] [**–f** *device*] [*field*[=*value*] . . .]

DESCRIPTION

eeprom displays or changes the values of fields in the EEPROM. It processes fields in the order given. When processing a *field* accompanied by a *value*, **eeprom** makes the indicated alteration to the EEPROM; otherwise it displays the *field*'s value. When given no field specifiers, **eeprom** displays the values of all EEPROM fields. A '–' flag specifies that fields and values are to be read from the standard input (one *field* or *field*=*value* per line).

Only the super-user may alter the EEPROM contents.

eeprom verifies the EEPROM checksums and complains if they are incorrect; if the **–i** flag is specified, erroneous checksums are ignored. If the **–c** flag is specified, all incorrect checksums are recomputed and corrected in the EEPROM.

The PROM monitor supports three security modes designated by the *secure* field: non-secure, command secure, and fully secure.

If *secure*=**none** the PROM monitor runs in the non-secure mode. In this mode all PROM monitor commands are allowed with no password required.

If **secure=command** the PROM monitor is in the command secure mode. In this mode, only the **b** (boot) command with no parameters and the **c** (continue) command with no parameters may be entered without a password being required. Any other command requires that the PROM monitor password be entered.

If **secure=full** the PROM monitor is in the fully secure mode. In this mode, only the **c** (continue) command with no parameters may be entered without a password being required. Entry of any other command requires that the PROM monitor password be entered. Note: the system will not auto-reboot in fully secure mode. The PROM monitor password must be entered before the boot process will take place.

When changing the security mode from non-secure to either command secure or fully secure, **eeprom** prompts for the entry and re-entry of a new PROM password as in the **passwd**(1) command. Changing from one secure mode to the other secure mode, or to the non-secure mode does not prompt for a password. Changing to non-secure mode erases the password.

The content of the **password** field is never displayed to any user. If the security mode is not **none**, the super-user may change the PROM monitor password by entering:

example# eeprom password=

eeprom prompts for a new password to be entered and re-entered.

The field **bad_login** maintains the count of bad login tries. It may be reset to zero (0) by specifying **bad_login=reset**.

OPTIONS

–c	Correct bad checksums. (Ignored on SPARCstation 1 systems.)
–i	Ignore bad checksums. (Ignored on SPARCstation 1 systems.)
–f *device*	Use *device* as the EEPROM device.

FIELDS and VALUES

hwupdate	a valid date (including **today** and **now**)
memsize	8 bit integer (megabytes of memory on machine)
memtest	8 bit integer (megabytes of memory to test)
scrsize	**1024x1024**, **1152x900**, **1600x1280**, or **1440x1440**

watchdog_reboot	**true** or **false**
default_boot	**true** or **false**
bootdev	*charchar*(*hex-int*,*hex-int*,*hex-int*) (with *char* a character, and *hex-int* a hexadecimal integer.)
kbdtype	8 bit integer (0 for all Sun keyboards)
keyclick	**true** or **false**
console	**b&w** or **ttya** or **ttyb** or **color**
custom_logo	**true** or **false**
banner	banner string
diagdev	%c%c (%x,%x,%x) — diagnostic boot device
diagpath	diagnostic boot path
ttya_no_rtsdtr	**true** or **false**
ttyb_no_rtsdtr	**true** or **false**
ttya_use_baud	**true** or **false**
ttyb_use_baud	**true** or **false**
ttya_baud	baud rate (16-bit decimal integer)
ttyb_baud	baud rate (16-bit decimal integer)
columns	number of columns on screen (8-bit integer)
rows	number of rows on screen (8-bit integer)
secure	**none**, **command**, or **full**
bad_login	number of bad login tries (16-bit unsigned integer, 0 if **reset**)
password	PROM monitor password (8-bytes)

FIELDS and VALUES — SPARCstation 1 SYSTEMS

hardware-revision	7 chars (for example, **30Mar88)**
selftest-#megs	32 bit decimal integer (megabytes of memory to test)
watchdog-reboot?	**true** or **false**; **true** to reboot after watchdog reset
boot-from	A string specifying boot string (for example, **le( )vmunix**); defaults to **vmunix**
keyboard-click?	**true** or **false**; **true** to enable clicking of keys on each keystroke
input-device	A string specifying one of **keyboard**, **ttya**, or **ttyb**; if the specified device is unavailable, **ttya** is used for both input and output *only* if input-device specified the keyboard *and* output-device specified the screen.
output-device	A string specifying one of **screen**, **ttya**, or **ttyb**; if the specified device is unavailable, **ttya** is used for *both* input and output *only* if input-device specified the keyboard *and* output-device specified the screen.
oem-banner?	**true** or **false**; **true** to use custom banner string instead of Sun banner
oem-banner	80 chars for custom banner string
oem-logo?	**true** or **false**; **true** to display custom logo instead of Sun logo
oem-logo	Name of file (in **iconedit** format) containing custom logo.
boot-from-diag	80 chars specifying diag boot string (for example, **sd()dexec**); defaults to **le( )vmunix**
ttya-mode	16 chars to specify 5 comma-separated fields of configuration information (for example, **1200,8,1,n,–**); defaults to **9600,8,1,n,–**. Fields, in left-to-right order, are: baud rate: 110, 300, 1200, 4800, 9600 . . . data bits: 5, 6, 7, 8 parity: n(none), e(even), o(odd), m(mark), s(space) stop bits: 1, 1.5, 2 handshake: –(none), h(hardware:rts/cts), s(software:xon/xoff)
ttyb-mode	16 chars to specify 5 comma-separated fields of configuration information (for example, **1200,7,1,n,s**); defaults to **9600,8,1,n,–**.

Fields, in left-to-right order, are:

baud rate:	110, 300, 1200, 4800, 9600 . . .
data bits:	5, 6, 7, 8
stop bits:	1, 1.5, 2
parity:	n(none), e(even), o(odd), m(mark), s(space)
handshake:	–(none), h(hardware:rts/cts), s(software:xon/xoff)

ttyb-rts-dtr-off **true** or **false**. Defaults to **false**.

ttya-rts-dtr-off **true** or **false**. Defaults to **false**.

ttya-ignore-cd **true** or **false**. Defaults to **true**.

ttyb-ignore-cd **true** or **false**; **true** to ignore the CARRIER DETECT line. Defaults to **true**.

screen-#rows number of rows on output device; defaults to 34 (for some devices actual values used may be less)

screen-#columns number of columns on output device; defaults to 80 (for some devices actual values used may be less)

auto-boot? **true** or **false**; **true** to boot on power-on

scsi-initiator-id An integer between 0 and 7 that specifies the SCSI initiator ID of the onboard SCSI host adapter.

sd-targets An array of 8 integers that map SCSI disk unit numbers to SCSI target numbers. The unit number is used to index into this string. The default settings are **31204567**, which means that unit 0 maps to target 3, unit 1 maps to target 1, and so on.

st-targets An array of 8 integers that map SCSI tape unit numbers to SCSI target numbers. The unit number is used to index into this string. The default settings are **45670123**, which means that unit 0 maps to target 4, unit 1 maps to target 5, and so on.

sunmon-compat? **true** or **false**. Defaults to **true**.

sbus-probe-list Defaults to 0123.

fcode-debug? **true** or **false**. Defaults to **false**.

last-hardware-update Date the CPU board was manufactured or upgraded to the latest hardware revision. The format is a human-readable date string, such as **23May89**.

testarea Defaults to 0.

mfg-switch? **true** or **false**. Defaults to **false**.

diag-switch? **true** or **false**. Defaults to **true**.

FILES

/dev/eeprom

FILES — SPARCstation 1 SYSTEMS

/dev/openprom

SEE ALSO

passwd(1)

PROM User's Manual

NAME

etherd, rpc.etherd – Ethernet statistics server

SYNOPSIS

/usr/etc/rpc.etherd *interface*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

etherd is a server which puts *interface* into promiscuous mode, and keeps summary statistics of all the packets received on that interface. It responds to RPC requests for the summary. You must be root to run **etherd**.

interface is a networking interface such as **ie0**, **ie1**, **ec0**, **ec1** and **le0**.

traffic(1C) displays the information obtained from **etherd** in graphical form.

SEE ALSO

traffic(1C)

NAME

etherfind – find packets on Ethernet

SYNOPSIS

etherfind [**–d**] [**–n**] [**–p**] [**–r**] [**–t**] [**–u**] [**–v**] [**–x**] [**–c** *count*] [**–i** *interface*] [**–l** *length*] *expression*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

etherfind prints out the information about packets on the ethernet that match the boolean *expression*. The short display, without the –v option, displays only the destination and src (with port numbers). When an Internet packet is fragmented into more than one ethernet packet, all fragments except the first are marked with an asterisk. With the –v option, the display is much more verbose, giving a trace that is suitable for analyzing many network problems. You must be root to invoke **etherfind**.

OPTIONS

–d Print the number of dropped packets. Not necessarily reliable.

–n Do not convert host addresses and port numbers to names.

–p Normally, the selected interface is put into promiscuous mode, so that **etherfind** has access to all packets on the ethernet. However, when the **–p** flag is used, the interface will not go promiscuous.

–r RPC mode: treat each packet as an RPC message, printing the program and procedure numbers. Routing packets are also more fully decoded using this option, and Network Interface Service (NIS) and NFS requests have their arguments printed.

–t Timestamps: precede each packet listing with a time value in seconds and hundredths of seconds since the first packet.

–u Make the output line buffered.

–v Verbose mode: print out some of the fields of TCP and UDP packets.

–x Dump the packet in hex, in addition to the line printed for each packet by default. Use the **–l** option to limit this printout.

–c *count*
Exit after receiving *count* packets. This is sometimes useful for dumping a sample of ethernet traffic to a file for later analysis.

–i *interface*
etherfind listens on *interface*. The program **netstat**(8C) when invoked with the **–i** flag lists all the interfaces that a machine has.

–l *length*
Use with the **–x** option to limit the number of bytes printed out.

expression
The syntax of *expression* is similar to that used by **find**(1). Here are the allowable primaries.

dst *destination*
True if the destination field of the packet is *destination*, which may be either an address or a name.

src *source*
True if the source field of the packet is *source*, which may be either an address or a name.

host *name*
True if either the source or the destination of the packet is *name*.

between *host1 host2*
True if either the source of the packet is *host1* and the destination *host2*, or the source is *host2* and the destination *host1*.

dstnet *destination*
True if the destination field of the packet has a network part of *destination*, which may be either an address or a name.

srcnet *source*
True if the source field of the packet has a network part of *source*, which may be either an address or a name.

srcport *port*
True if the packet has a source port value of *port*. This will check the source port value of either UDP or TCP packets (see **tcp**(4P)), and **udp**(4P)). The *port* can be a number or a name used in **/etc/services**.

dstport *port*
True if the packet has a destination port value of *port*. The *port* can be a number or a name.

less *length*
True if the packet has a length less than or equal to *length*.

greater *length*
True if the packet has a length greater than or equal to *length*.

–proto *protocol*
True if the packet is an IP packet (see **ip**(4P)) of protocol type *protocol*. *Protocol* can be a number or one of the names **icmp**, **udp**, **nd**, or **tcp**.

byte *byte op value*
True if byte number *byte* of the packet is in relation *op* to *value*. Legal values for *op* are **+**, **<**, **>**, **&**, and **|**. Thus **4=6** is true if the fourth byte of the packet has the value 6, and **20&0xf** is true if byte twenty has one of its four low order bits nonzero.

broadcast
True if the packet is a broadcast packet.

arp True if the packet is an ARP packet (see **arp**(4P)).

rarp True if the packet is a rarp packet.

–ip True if the packet is an IP packet.

–decnet
True if the packet is a DECNET packet.

–apple True if the packet is an AppleTalk protocol packet.

The primaries may be combined using the following operators (in order of decreasing precedence):

A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

The negation of a primary ('**not**' is the unary *not* operator).

Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries, or can be specified with '**and**').

Alternation of primaries ('**or**' is the *or* operator).

EXAMPLE

To find all packets arriving at or departing from the host **sundown**, or that are ICMP packets:

example% etherfind host sundown or proto icmp

SEE ALSO

find(1), **traffic**(1C), **arp**(4P), **ip**(4P), **nit**(4P) **tcp**(4P), **udp**(4P), **netstat**(8C)

BUGS

The syntax is painful.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

exportfs – export and unexport directories to NFS clients

SYNOPSIS

/usr/etc/exportfs [**–aiuv**] [**–o** *options*] [*pathname*]

DESCRIPTION

exportfs makes a local directory or filename available for mounting over the network by NFS clients. It is normally invoked at boot time by the **/etc/rc.local** script, and uses information contained in the **/etc/exports** file to export *pathname* (which must be specified as a full pathname). The super-user can run **exportfs** at any time to alter the list or characteristics of exported directories and filenames. Directories and files that are currently exported are listed in the file **/etc/xtab.**

With no options or arguments, **exportfs** prints out the list of directories and filenames currently exported.

OPTIONS

–a All. Export all pathnames listed in **/etc/exports,** or if **–u** is specified, unexport all of the currently exported pathnames.

–i Ignore the options in **/etc/exports.** Normally, **exportfs** will consult **/etc/exports** for the options associated with the exported pathname.

–u Unexport the indicated pathnames.

–v Verbose. Print each directory or filename as it is exported or unexported.

–o *options*
Specify a comma-separated list of optional characteristics for the pathname being exported. *options* can be selected from among:

ro Export the pathname read-only. If not specified, the pathname is exported read-write.

rw=*hostname*[**:***hostname*]...
Export the pathname read-mostly. Read-mostly means exported read-only to most machines, but read-write to those specified. If not specified, the pathname is exported read-write to all.

anon=*uid*
If a request comes from an unknown user, use UID as the effective user ID. Note: root users (UID 0) are always considered "unknown" by the NFS server, unless they are included in the **root** option below. The default value for this option is –2. Setting the value of "anon" to –1 disables anonymous access. Note: by default secure NFS accepts insecure requests as anonymous, and those wishing for extra security can disable this feature by setting "anon" to –1.

root=*hostname*[**:***hostname*]...
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

access=*client*[**:***client*]...
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see **netgroup**(5)). Each *client* in the list is first checked for in the **/etc/netgroup** database, and then the **/etc/hosts** database. The default value allows any machine to mount the given directory.

secure Require clients to use a more secure protocol when accessing the directory.

FILES

/etc/exports	static export information
/etc/xtab	current state of exported pathnames
/etc/netgroup	

SEE ALSO

exports(5), **netgroup**(5), **showmount**(8)

WARNINGS

You cannot export a directory that is either a parent- or a sub-directory of one that is currently exported and *within the same filesystem*. It would be illegal, for example, to export both **/usr** and **/usr/local** if both directories resided in the same disk partition.

NAME

extract_patch – extract and execute patch files from installation tapes

SYNOPSIS

extract_patch [–**d***device* [–**r***remote-host*]] [–**p***patch-name*] [–**DEFAULT**]

DESCRIPTION

extract_patch extracts a patch from a release tape onto the current system. If no options are specifed, it prompts for input as to the patch name, tape device, or remote hostname from which to the software is to be installed. If the named patch cannot be found, a list of valid patches are printed.

If the named patch is found then the patch is extracted from the tape onto the system. If there is a **README** file in the extracted contents then the user is given a chance to view it. If there is a patch installation program the user is given a chance to run it.

Patches must appear in the tape's table of contents, and must have a name that starts with "Patch_".

OPTIONS

–**d***device*
: Install from the indicated tape drive, such as **st0**, or **mt0**.

–**r***remote-host*
: Install from the device given in the –**d** option on the indicated remote host.

–**p***patch-name*
: Specifes the name of the patch to extract.

–**DEFAULT**
: Execute the installation script using all default values. Otherwise the installation script prompts for any optional values.

SEE ALSO

extract_unbundled(8)

NAME

extract_unbundled – extract and execute unbundled-product installation scripts

SYNOPSIS

extract_unbundled [–**d***device* [–**r***remote-host*]] [**–DEFAULT**]

DESCRIPTION

extract_unbundled extracts and executes the installation scripts from release tapes for Sun unbundled software products. If no options are specified, it prompts for input as to the tape device, or remote host-name from which to the software is to be installed. For information about installing a specific product, refer to the installation manual that accompanies that product.

OPTIONS

–**d***device*
: Install from the indicated tape drive, such as **st0** or **mt0**.

–**r***remote_host*
: Install from the device given in the –**d** option on the indicated remote host.

–DEFAULT
: Execute the installation script using all default values. Otherwise the installation script prompts for any optional values.

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/usr/etc/fastboot [*boot-options*]

/usr/etc/fasthalt [*halt-options*]

DESCRIPTION

fastboot and **fasthalt** are shell scripts that reboot and halt the system without checking the file systems. This is done by creating a file **/fastboot**, then invoking the **reboot**(8) program. The system startup script, **/etc/rc**, looks for this file and, if present, skips the normal invocation of **fsck**(8).

FILES

/usr/etc/fastboot
/etc/rc

SEE ALSO

fsck(8), **halt**(8), **init**(8), **rc**(8), **reboot**(8)

NAME

fingerd, in.fingerd – remote user information server

SYNOPSIS

/usr/etc/in.fingerd

DESCRIPTION

fingerd implements the server side of the Name/Finger protocol, specified in RFC 742. The Name/Finger protocol provides a remote interface to programs which display information on system status and individual users. The protocol imposes little structure on the format of the exchange between client and server. The client provides a single "command line" to the finger server which returns a printable reply.

fingerd waits for connections on TCP port 79. Once connected it reads a single command line terminated by a LINEFEED which is passed to **finger**(1). **fingerd** closes its connections as soon as the output is finished.

If the line is null (only a LINEFEED is sent) then **finger** returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (for instance, ericLINEFEED) then the response lists more extended information for only that particular user, whether logged in or not. Allowable "names" in the command line include both "login names" and "user names". If a name is ambiguous, all possible derivations are returned.

SEE ALSO

finger(1)

Harrenstien, Ken, *NAME/FINGER*, RFC 742, Network Information Center, SRI International, Menlo Park, Calif., December 1977.

BUGS

Connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. **fingerd** should be taught to filter out IAC's and perhaps even respond negatively (IAC *will not)* to all option commands received.

NAME

fonftlip – create Sun386i-style vfont file

SYNOPSIS

fontflip fontname [**–o** *newfontname*]

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

fontflip takes as input a vfont file (Sun-3 fixedwidthfont) and creates a Sun386i system vfont. This new font is a bitflipped version of its input. The new font is named *oldfont*.**flip** unless otherwise specified.

OPTIONS

–o *newfontname* Specify the name of the new flipped font.

FILES

/usr/lib/fonts/fixedwidthfonts

SEE ALSO

vfont(5)

NAME

format – disk partitioning and maintenance utility

SYNOPSIS

format [**–f** *command-file*] [**–l** *log-file*] [**–x** *data-file*] [**–d** *disk-name*] [**–t** *disk_type*] [**–p** *partition-name*] [**–s**] *diskname*. . .

DESCRIPTION

format enables you to format, label, repair and analyze disks on your Sun computer. Unlike previous disk maintenance programs, **format** runs under SunOS. Because there are limitations to what can be done to the system disk while the system is running, **format** is also supported within the memory-resident system environment. For most applications, however, running **format** under SunOS is the more convenient approach.

If no *disk-list* is present, **format** uses the disk list defined in the data file specified with the **–x** option. If that option is omitted, the data file defaults to **format.dat** in the current directory, or else **/etc/format.dat**.

OPTIONS

–f *command-file*
: Take command input from *command-file* rather than the standard input. The file must contain commands that appear just as they would if they had been entered from the keyboard. With this option, **format** does not issue **continue?** prompts.

–l *log-file*
: Log a transcript of the **format** session to the indicated *log-file*, including the standard input, the standard output and the standard error.

–x *data-file*
: Use the disk list contained in *data-file*.

–d *disk_name*
: Specify which disk should be made current upon entry into the program. The disk is specified by its logical name (for instance, - xy0). This can also be accomplished by specifying a single disk in the disk list.

–t *disk-type*
: Specify the type of disk which is current upon entry into the program, A disk's type is specified by name in the data file. This option can only be used if a disk is being made current as described above.

–p *partition-name*
: Specify the partition table for the disk which is current upon entry into the program. The table is specified by its name as defined in the data file. This option can only be used if a disk is being made current, and its type is either specified or available from the disk label.

–s
: Silent. Suppress all of the standard output. Error messages are still displayed. This is generally used in conjunction with the **–f** option.

FILES

/etc/format.dat default data file

SEE ALSO

System and Network Administration

NAME

fpa_download – download to the Floating Point Accelerator

SYNOPSIS

fpa_download [**–d**] [–r] [–v] [**–u** *ufile*] [**–m** *mfile*] [**–c** *cfile*]

AVAILABILITY

fpa_download applies to Sun-3 and Sun-3x systems equipped with either an FPA or FPA+.

DESCRIPTION

fpa_download writes microcode, map, and constants files to FPA and FPA+ boards. FPA requires a map file; FPA+ does not.

Root execution level is required to download (d,u,m and c options). **fpa_download** is called from **/etc/rc.local** when **/dev/fpa** exists.

Given no arguments, **fpa_download** prints whether an FPA, or FPA+ is installed.

OPTIONS

–d Download microcode, constants, and map files. Enable default file names.

–r Print microcode and constant revision.

–v Verbose mode.

–u *ufile* Download microcode from *ufile*.

–m *mfile* Download map from *mfile* (FPA only).

–c *cfile* Download constants from *cfile*.

FILES

/dev/fpa	device file for both FPA and FPA+.
/usr/etc/fpa/fpa_micro_bin	default microcode file (*ufile*) for FPA.
/usr/etc/fpa/fpa_constants	default constants file (cfile) for FPA
/usr/etc/fpa/fpa_micro_map	default map file (mfile) for FPA
/usr/etc/fpa/fpa_micro_bin+	default microcode file (ufile) for FPA+
/usr/etc/fpa/fpa_constants+	default constants file (cfile) for FPA+

SEE ALSO

fpa(4)

DIAGNOSTICS

The following diagnostics are printed when **fpa_download** encounters a serious error and asks the kernel to disable the FPA. This might occur if the microcode, map, or constants files are corrupted, or if there is an FPA or system hardware problem.

FPA Download Failed - FPA ioctl failed
An ioctl() on **/dev/fpa** failed, possibly due to a hung FPA pipe.

FPA Failed Download - FPA Bus Error
Received a SIGFPE.

FPA Failed Download - Upload mismatch
After each file is written to the FPA/FPA+, **fpa_download** uploads the contents of FPA memory and compares it with the source. They should always match.

NAME

fparel – Sun FPA online reliability tests

SYNOPSIS

fparel [**–p***n*] [**–v**]

AVAILABILITY

Not available on Sun386i systems.

DESCRIPTION

fparel is a command to execute the Sun FPA online confidence and reliability test program. **fparel** tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. **fparel** runs without disturbing other processes that may be using the FPA. **fparel** can only be run by the super-user.

After a successful pass, **fparel** writes

> **time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31**

to the file **/var/adm/diaglog**.

If a pass fails, **fparel** writes

> **time, date: Sun FPA failed**

along with the test name and context number that failed, to the file **/var/adm/diaglog**. **fparel** then broadcasts the message

> **time, date: Sun FPA failed, disabled, service required**

to all users of the system. Next, **fparel** causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file **/etc/rc.local** should contain an entry to cause **fparel** to be invoked upon reboot to be sure that the FPA remains unaccessible in cases where rebooting doesn't correct the problem. See **rc**(8).

The **crontab**(5) file for root should contain an entry indicating that **cron**(8) is to run **fparel** daily, such as:

> **7 2 * * * /usr/etc/fpa/fparel**

which causes **fparel** to run at seven minutes past two, every day. See **cron**(8) and **crontab**(5) for details.

OPTIONS

–p*n* Perform *n* passes. Default is *n*=1. **–p0** means perform 2147483647 passes.

–v Run in verbose mode with detailed test results to the standard output.

FILES

/var/adm/diaglog Log of **fparel** diagnostics.
/etc/rc.local
/var/spool/cron/crontabs/root
/usr/etc/fpa/* directory containing FPA microcode, data files, and loader

SEE ALSO

crontab(5), **cron**(8), **fpaversion**(8), **rc**(8)

NAME

fpaversion – print FPA version, load microcode

SYNOPSIS

fpaversion [**–chlqv**] [**–t** [**cdhimprstvxCIMS**]]

AVAILABILITY

Available only on Sun-3 and Sun-3x systems equipped with either an FPA or an FPA+.

DESCRIPTION

fpaversion performs various tests on the FPA or FPA+. Without arguments, it prints the microcode version number and constants currently installed on **/dev/fpa**. **fpaversion** also performs a quick test to ensure proper operation and reports whether an FPA or an FPA+ is installed.

OPTIONS

–c Continue tests after an error.

–h Help. Print command-line summary.

–l Loop through tests infinitely.

–q Quiet output. Print out only error messages.

–v Verbose output.

–t Specify certain tests:

- **c** Command register format instructions.
- **d** Double precision format instructions.
- **h** Help. Print summary of test specifiers.
- **i** Imask register.
- **m** Mode register.
- **p** Simple pipe sequencing.
- **r** User registers for all contexts.
- **s** Single precision format instructions.
- **t** Status generation.
- **v** Print version number and date of microcode, and constants. Report whether an FPA or an FPA+ is installed.
- **x** Extended format instructions.
- **C** Check checksum for microcode, mapping RAM, and constant RAM for the FPA. Check checksum for microcode RAM and constant RAM for the FPA+.
- **I** Allows interactive reads and writes to the FPA.
- **M** Command register format matrix instructions.
- **S** Shadow registers.

FILES

/dev/fpa	physical FPA device
/usr/etc/fpa/fpa_micro_bin	microcode binaries for the FPA
/usr/etc/fpa/fpa_micro_map	microcode map binaries for the FPA
/usr/etc/fpa/fpa_constants	microcode data file for the FPA
/usr/etc/fpa/fpa_micro_bin+	microcode binaries for the FPA+
/usr/etc/fpa/fpa_constants+	microcode data file for the FPA+
/usr/etc/fpa/fpa_download	microcode loader

SEE ALSO

fpa_download(8), **fparel**(8), **sundiag**(8)

DIAGNOSTICS

If a test fails, its name, along with the actual and expected results will be printed.

NAME

fpurel – perform tests the Sun Floating Point Co-processor.

SYNOPSIS

fpurel [**−v**] [**−p**[*count*]] [**−r**]

DESCRIPTION

fpurel performs a series of functional and computational tests for the Sun Floating Point Co-processor to verify that it is operational and accurate. With no options, **fpurel** runs one pass silently in the foreground and only reports errors if any are found.

OPTIONS

−v Verbose. Display the name and results of each test on the console. The default is to run silently.

−p[*count*]
Passcount. Specify the number of times to run the test suite. The default is to run one pass.

−r Disable stop on error. Continue to run if errors are detected. The default is to display the error message and to stop testing when an error is detected.

EXAMPLE

This example uses **fpurel** from the **/usr/diag** directory. If no errors are detected, then no information is displayed.

% /usr/diag/fpurel

NAME
fpuversion4 – print the Sun-4 FPU version

SYNOPSIS
/usr/etc/fpuversion4

AVAILABILITY
Sun-4 systems only.

DESCRIPTION
fpuversion4 reads the **%fsr** register to determine the FPU version installed on a Sun-4. The printed version field contains a value in the range 0-7; by SPARC convention 7 indicates that no FPU is installed, so floating-point instructions are always emulated in the kernel.

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/usr/etc/fsck –p [*filesystem* ...]

/usr/etc/fsck [**–b** *block#*] [**–w**] [**–y**] [**–n**] [**–c**] [*filesystem*] ...

DESCRIPTION

The first form of **fsck** preens a standard set of file systems or the specified file systems. It is normally used in the **/etc/rc** script during automatic reboot. In this case, **fsck** reads the table **/etc/fstab** to determine the file systems to check. It inspects disks in parallel, taking maximum advantage of I/O overlap to check the file systems as quickly as possible.

Normally, the root file system is checked in pass 1; other root-partition file systems are checked in pass 2. Small file systems on separate partitions are checked in pass 3, while larger ones are checked in passes 4 and 5.

Only partitions marked in **/etc/fstab** with a file system type of “4.2” and a non-zero pass number are checked.

fsck corrects innocuous inconsistencies such as: unreferenced inodes, too-large link counts in inodes, missing blocks in the free list, blocks appearing in the free list and also in files, or incorrect counts in the super block, automatically. It displays a message for each inconsistency corrected that identifies the nature of, and file system on which, the correction is to take place. After successfully correcting a file system, **fsck** prints the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If **fsck** encounters other inconsistencies that it cannot fix automatically, it exits with an abnormal return status (and the reboot fails).

If sent a QUIT signal, **fsck** will finish the file system checks, then exit with an abnormal return status that causes the automatic reboot to fail. This is useful when you wish to finish the file system checks, but do not want the machine to come up multiuser.

Without the **–p** option, **fsck** audits and interactively repairs inconsistent conditions on file systems. In this case, it asks for confirmation before attempting any corrections. Inconsistencies other than those mentioned above can often result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output.

The default action for each correction is to wait for the operator to respond either **yes** or **no**. If the operator does not have write permission on the file system, **fsck** will default to a **–n** (no corrections) action.

If no file systems are given to **fsck** then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked in order are as follows:

- Blocks claimed by more than one inode or the free list.
- Blocks claimed by an inode or the free list outside the range of the file system.
- Incorrect link counts.
- Incorrect directory sizes.
- Bad inode format.
- Blocks not accounted for anywhere.
- Directory checks, file pointing to unallocated inode, inode number out of range.
- Super Block checks: more blocks for inodes than there are in the file system.
- Bad free block list format.
- Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator’s concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. If the **lost+found** directory does not exist, it is created. If there is insufficient space its size is increased.

A file system may be specified by giving the name of the cooked or raw device on which it resides, or by giving the name of its mount point. If the latter is given, **fsck** finds the name of the device on which the file system resides by looking in **/etc/fstab**.

Checking the raw device is almost always faster.

OPTIONS

–b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

–w Check writable file systems only.

–y Assume a **yes** response to all questions asked by **fsck**; this should be used with extreme caution, as it is a free license to continue, even after severe problems are encountered.

–n Assume a **no** response to all questions asked by **fsck**; do not open the file system for writing.

–c If the file system is in the old (static table) format, convert it to the new (dynamic table) format. If the file system is in the new format, convert it to the old format provided the old format can support the filesystem configuration. In interactive mode, **fsck** will list the direction the conversion is to be made and ask whether the conversion should be done. If a negative answer is given, no further operations are done on the filesystem. In preen mode, the direction of the conversion is listed and done if possible without user interaction. Conversion in preen mode is best used when all the file systems are being converted at once. The format of a file system can be determined from the first line of output from **dumpfs**(8)

FILES

/etc/fstab default list of file systems to check

DIAGNOSTICS

The diagnostics produced by **fsck** are fully enumerated and explained in *System and Network Administration.*

EXIT STATUS

0 Either no errors detected or all errors were corrected.

4 Root file system errors were corrected. The system must be rebooted.

8 Some uncorrected errors exist on one or more of the file systems checked, there was a syntax error, or some other operational error occurred.

12 A signal was caught during processing.

SEE ALSO

fs(5), **fstab**(5), **dumpfs**(8), **newfs**(8), **mkfs**(8), **panic**(8S), **reboot**(8), **rexecd**(8C), **ypserv**(8)

System and Network Administration

BUGS

There should be some way to start a '**fsck –p**' at pass *n*.

NAME

fsirand – install random inode generation numbers

SYNOPSIS

fsirand [**−p**] *special*

DESCRIPTION

fsirand installs random inode generation numbers on all the inodes on device *special*, and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

fsirand must be used only on an unmounted filesystem that has been checked with **fsck**(8). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately rebooted afterwords.

OPTIONS

−p Print out the generation numbers for all the inodes, but do not change the generation numbers.

SEE ALSO

fsck(8)

NAME

ftpd, in.ftpd – TCP/IP Internet File Transfer Protocol server

SYNOPSIS

/usr/etc/in.ftpd [**–dl**] [**–t***timeout*] *host.socket*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

ftpd is the TCP/IP Internet File Transfer Protocol (FTP) server process. The server is invoked by the Internet daemon **inetd**(8C) each time a connection to the FTP service (see **services**(5)) is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 60 seconds.

If the **–d** option is specified, debugging information is logged to the system log daemon, **syslogd**(8).

If the **–l** option is specified, each FTP session is logged to **syslogd**.

The FTP server will timeout an inactive session after 15 minutes. If the **–t** option is specified, the inactivity timeout period will be set to *timeout*.

The FTP server currently supports the following FTP requests; case is not distinguished.

Request	**Description**
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (**ls –lg**)
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory (**ls**)
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name

STOR store a file

STOU store a file with a unique name

STRU specify data transfer *structure*

TYPE specify data transfer *type*

USER specify user name

XCUP change to parent of current working directory

XCWD change working directory

XMKD make a directory

XPWD print the current working directory

XRMD remove a directory

The remaining FTP requests specified in RFC 959 are recognized, but not implemented.

The FTP server will abort an active file transfer only when the **ABOR** command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in RFC 959.

ftpd interprets file names according to the "globbing" conventions used by **csh**(1). This allows users to utilize the metacharacters '* ? [] {} ~'.

ftpd authenticates users according to three rules.

- The user name must be in the password data base, **/etc/passwd,** and not have a null password. In this case a password must be provided by the client before any file operations may be performed.
- If the file **/etc/ftpusers** exists, the user name must not appear in that file.
- The user must have a standard shell returned by **getusershell**(3).
- If the user name is "anonymous" or "ftp", an anonymous FTP account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, **ftpd** takes special measures to restrict the client's access privileges. The server performs a **chroot**(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin Make this directory owned by the super-user and unwritable by anyone. The program **ls**(1V) must be present to support the list commands. This program should have mode 111. Since the default /bin/ls command is linked with a shared library, so you need to set up the files for dynamic linking as well.

~ftp/usr/lib/ld.so
the runtime loader must be present and executable.

~ftp/dev/zero
used by the runtime loader, create this with the command "mknod zero c 3 12".

~ftp/usr/lib/libc.so.*
should be a copy of the latest version of the shared C library.

~ftp/etc Make this directory owned by the super-user and unwritable by anyone. The files **passwd**(5) and **group**(5) must be present for the **ls** command to work properly. These files should be mode 444.

~ftp/pub Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

DIAGNOSTICS

ftpd logs various errors to the system log daemon, **syslogd**, with a facility code of **daemon**. The messages are listed here, grouped by severity level.

Err Severity

getpeername failed: *reason*
A **getpeername**(2) call failed.

getsockname failed: *reason*
A **getsockname**(2) call failed.

signal failed: *reason*
A **signal** (3V) (see **signal**(3V)) call failed.

setsockopt failed: *reason*
A **setsockopt** call (see **getsockopt**(2)) failed.

ioctl failed: *reason*
A **ioctl**(2) call failed.

directory: *reason*
ftpd did not have write permission on the directory *directory* in which a file was to be created by the **STOU** command.

Info Severity

These messages are logged only if the **–l** flag is specified.

FTPD: connection from *host* **at** *time*
A connection was made to **ftpd** from the host *host* at the date and time *time*.

FTPD: User *user* **timed out after** *timeout* **seconds at** *time*
The user *user* was logged out because they hadn't entered any commands after *timeout* seconds; the logout occurred at the date and time *time*.

Debug Severity

These messages are logged only if the **–d** flag is specified.

TPD: command: *command*
A command line containing *command* was read from the FTP client.

lost connection
The FTP client dropped the connection.

<——— *replycode*
<——— *replycode–*
A reply was sent to the FTP client with the reply code *replycode*. The next message logged will include the message associated with the reply. If a – follows the reply code, the reply is continued on later lines.

SEE ALSO

csh(1), **ftp**(1C), **ls**(1V), **chroot**(2) **getpeername**(2), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **getusershell**(3), **ftpusers**(5), **group**(5), **passwd**(5), **services**(5), **inetd**(8C), **syslogd**(8)

Postel, Jon, and Joyce Reynolds, *File Transfer Protocol (FTP)*, RFC 959, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

BUGS

The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user ID of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

fumount – force unmount of an advertised RFS resource

SYNOPSIS

fumount [**–w** *seconds*] *resource*

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

fumount unadvertises *resource* and disconnects remote access to the resource.

When the forced unmount occurs, an administrative shell script, **rfuadmin**, is started on each remote system that has the resource mounted If a grace period is specified (in seconds), **rfuadmin**(8) is started with the **fuwarn** option. When the actual forced unmount is ready to occur, **rfuadmin**(8) is started with the **fumount** option. See **rfuadmin**(8) for information on the action taken in response to the forced unmount.

This command is restricted to the super-user.

An error message will be sent to standard error if any of the following are true of *resource*:

- It does not physically reside on the local machine.
- It is an invalid resource name.
- It is not currently advertised and is not remotely mounted.

OPTION

–w *seconds* Delay execution of the disconnect *seconds* seconds.

SEE ALSO

adv(8), **mount**(8), **rfuadmin**(8), **rfudaemon**(8), **unadv**(8)

NAME

fusage – RFS disk access profiler

SYNOPSIS

fusage [[*mount_point*] | [*advertised_resource*] | [*block_special_device*] [. . .]]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

When used with no options, **fusage** reports block I/O transfers, in kilobytes, to and from all locally mounted file systems and advertised Remote File Sharing resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, **fusage** reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

df(1V), **adv**(8), **crash**(8), **mount**(8)

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/usr/etc/fuser [**–ku**] *filename* | *resource* [–] [[**–ku**] *filename* | *resource*]

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *filename*s or remote *resources* specified as arguments. Each process ID is followed by a letter code. Possible code letters and an explanation of how the process is using the file are given below:

- **c** its current directory
- **p** the parent of its current directory (only when the file is being used by the system)
- **r** its root directory
- **v** process has **exec**'ed or **mmap**'ed file

For block special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource are reported. **fuser** cannot use the mount point of the remote resource to report all processes using any file associated with that remote resource; it must use the resource name. For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The process IDs are printed as a single line on the standard output, separated by SPACE characters and terminated with a single NEWLINE. All other output is written on standard error.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use **fuser**.

Only the super-user can terminate another user's process

OPTIONS

If more than one group of files are specified, the options may be respecified for each additional group of files.

- **–** Cancel the options currently in force. The new set of options applies to the next group of files.
- **–k** Send SIGKILL signal to each process. Since this option spawns kills for each process, the kill messages may not show up immediately (see **kill**(2V)).
- **–u** User login name, in parentheses, also follows the process ID.

FILES

/vmunix	system namelist
/dev/kmem	system image
/dev/mem	system image

SEE ALSO

ps(1), **kill**(2V), **signal**(3V), **mount**(8)

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**–ci**]

/usr/lib/acct/wtmpfix [*filename* ...]

DESCRIPTION

fwtmp

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing bad records, using a text editor, or general purpose maintenance of the file.

wtmpfix

wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A '–' can be used in place of *filename* to indicate the standard input. If time/date corrections are not performed, **acctcon1** fails when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/var/adm/wtmp**. The first record is the old date denoted by the string '|' placed in the line field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string '{' placed in the line field. **wtmpfix** uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, **wtmpfix** checks the validity of the name field to ensure that it consists solely of alphanumeric characters or SPACE characters. If it encounters a name that is considered invalid, it changes the login name to **INVALID** and writes a diagnostic message to the standard error. In this way, **wtmpfix** reduces the chance that **acctcon1** will fail when processing connect accounting records.

OPTIONS

fwtmp

–c Write output in binary form.

–i Input is in ASCII form.

FILES

/var/adm/wtmp

SEE ALSO

acctcom(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **runacct**(8)

NAME

gettable – get DARPA Internet format host table from a host

SYNOPSIS

/usr/etc/gettable *host*

DESCRIPTION

gettable is a simple program used to obtain the DARPA Internet host table from a "hostname" server. The indicated *host* is queried for the table. The table, if retrieved, is placed in the file **hosts.txt**.

gettable operates by opening a TCP connection to the port indicated in the service specification for "hostname" . A request is then made for "ALL" names and the resultant information is placed in the output file.

gettable is best used in conjunction with the **htable**(8) program which converts the DARPA Internet host table format to that used by the network library lookup routines.

SEE ALSO

intro(3), htable(8)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *HOSTNAME Server,* RFC 953, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

BUGS

Should allow requests for only part of the database.

NAME

getty – set terminal mode

SYNOPSIS

/usr/etc/getty [*type* [*tty*]]

Sun386i SYSTEM SYNOPSIS

/usr/etc/getty [**–n**] [*type* [*tty*]]

DESCRIPTION

getty, which is invoked by **init**(8), opens and initializes a tty line, reads a login name, and invokes **login**(1).

The *tty* argument is the name of the character-special file in **/dev** that corresponds to the terminal. If there is no *tty* argument, or the argument is '–', the tty line is assumed to be opened as file descriptor **0**.

The *type* argument, if supplied, is used as an index into the **gettytab**(5) database—to determine the characteristics of the line. If this argument is absent, or if there is no such entry, the default entry is used. If there is no **/etc/gettytab** file, a set of system-supplied defaults is used.

When the indicated entry is located, **getty** clears the terminal screen, prints a banner heading, and prompts for a login name. Usually, either the banner or the login prompt includes the system's hostname.

Next, **getty** prompts for a login and reads the login name, one character at a time. When it receives a null character (which is assumed to be the result pressing the BREAK , or "interrupt" key), **getty** switches to the entry **gettytab** entry named in the **nx** field. It reinitializes the line to the new characteristics, and then prompts for a login once again. This mechanism typically is used to cycle through a set of line speeds (baud rates) for each terminal line. For instance, a rotary dialup might have entries for the speeds: 300, 1200, 150, and 110 baud, with each **nx** field pointing to the next one in succession.

The user terminates login input line with a NEWLINE or RETURN character. The latter is preferable; it sets up the proper treatment of RETURN characters (see **tty**(4)). **getty** checks to see if the terminal has only upper-case alphabetical characters. If all alphabetical characters in the login name are in upper case, the system maps them along with all subsequent upper-case input characters to lower-case internally; they are displayed in upper case for the benefit of the terminal. To force recognition of an upper-case character, the shell allows them to be quoted (typically by preceding each with a backslash, '\').

Finally, **getty** calls **login**(1) with the login name as an argument.

getty can be set to time out after a certain interval; this hangs up dial-up lines if the login name is not entered in time.

Sun386i SYSTEM DESCRIPTION

For Sun386i system, the value of *type* is the constant **Sun**, for the console frame buffer.

Sun386i SYSTEM OPTIONS

–n invoke the full screen login program **logintool**(8), and optionally the "New User Accounts" feature. May only be used on a frame buffer. Unless removed from the console entry in */etc/ttytab*, this option is in effect by default.

FILES

/etc/gettytab

SEE ALSO

login(1), **ioctl**(2), **tty**(4), **fbtab**(5), **gettytab**(5), **svdtab**(5), **ttytab**(5), **init**(8), **logintool**(8)

DIAGNOSTICS

ttyxx: **No such device or address.**

ttyxx: **No such file or directory.**

A terminal which is turned on in the **ttys** file cannot be opened, likely because the requisite lines are either not configured into the system, the associated device was not attached during boot-time system configuration, or the special file in **/dev** does not exist.

NAME

gpconfig – initialize the Graphics Processor

SYOPNSIS

/usr/etc/gpconfig *gpunit* [[**–b**] [**–f**] *fbunit*... [**–u** *microcode-file*]]

DESCRIPTION

gpconfig binds **cgtwo** frame buffers to the GP, (Graphics Processor) and loads and starts the appropriate microcode in the GP. For example, the command line:

/usr/etc/gpconfig gpone0 cgtwo0 cgtwo1

will bind the frame buffer boards **cgtwo0** and **cgtwo1** to the Graphics Processor **gpone0**. The devices **/dev/gpone0a** and **/dev/gpone0b** will then refer to the combination of **gpone** and **cgtwo0** or **cgtwo1** respectively.

The same **cgtwo** frame buffer cannot be bound to more than one GP.

All **cgtwo** frame buffer boards bound to a GP must be configured to the same width and height.

The standard version of the file **/etc/rc.local** contains the following **gpconfig** command line:

/usr/etc/gpconfig gpone0 –f –b cgtwo0

This binds **gpone0** and **cgtwo0** as **gpone0a**, causes **gpone0a** to use the Graphics Buffer Board if it is present, and redirects **/dev/fb** to be **/dev/gpone0a**. If another configuration is desired, edit the command line in **/etc/rc.local** to do the appropriate thing.

It is inadvisable to run the **gpconfig** command while the GP is being used. Unpredictable results may occur. If it is necessary to change the frame buffer bindings to the GP (or to stop using the GP altogether), bring the system down gently, boot single user, edit the **gpconfig** line in the **/etc/rc.local** file, and bring the system back up multiuser.

OPTIONS

–b Configure the GP to use the Graphics Buffer as well. Currently only one GP-to-frame-buffer binding is allowed to use the graphics buffer at a time. Only the last **–b** option in the command line takes effect.

–f Redirect **/dev/fb** to the device formed by binding *gpunit* with **fbunit**. Only the last **–f** option in the command line takes effect.

–u *microcode-file*
Load the specified microcode file instead of the default file from **/usr/lib**.

FILES

/dev/cgtwo[0-9]
/dev/fb
/dev/gpone[0-3][abcd]
/usr/lib/gp1cg2.1024.ucode
/usr/lib/gp1cg2.1152.ucode
/etc/rc.local

SEE ALSO

cgtwo(4S), **gpone**(4S)

NAME

grpck – check group database entries

SYNOPSIS

/usr/etc/grpck [*filename*]

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

grpck checks that a file in **group**(5) does not contain any errors; it checks the **/etc/group** file by default.

FILES

/etc/group

DIAGNOSTICS

Too many/few fields
An entry in the group file does not have the proper number of fields.

No group name
The group name field of an entry is empty.

Bad character(s) in group name
The group name in an entry contains characters other than lower-case letters and digits.

Invalid GID
The group ID field in an entry is not numeric or is greater than 65535.

Null login name
A login name in the list of login names in an entry is null.

Login name not found in password file
A login name in the list of login names in an entry is not in the password file.

First char in group name not lower case alpha
The group name in an entry does not begin witha lower-case letter.

Group name too long
The group name in an entry has more than 8 characters.

SEE ALSO

groups(1), **group**(5), **passwd**(5)

NAME

gxtest – stand alone test for the Sun video graphics board

SYNOPSIS

b /stand/gxtest

DESCRIPTION

gxtest runs stand alone, not under control of the operating system. With the PROM resident monitor in control of the system, type the command:

> b /stand/gxtest

and the monitor boots the video test program into memory. **gxtest** is completely self-explanatory and runs under its own steam. It reports any errors it finds on the screen.

NAME

halt – stop the processor

SYNOPSIS

/usr/etc/halt [**–nqy**]

DESCRIPTION

halt writes out any information pending to the disks and then stops the processor.

halt normally logs the system shutdown to the system log daemon, **syslogd**(8), and places a shutdown record in the login accounting file **/var/adm/wtmp**. These actions are inhibited if the **–n** or **–q** options are present.

OPTIONS

–n Prevent the *sync* before stopping.

–q Do a quick halt. No graceful shutdown is attempted.

–y Halt the system, even from a dialup terminal.

FILES

/var/adm/wtmp login accounting file

SEE ALSO

reboot(8), **shutdown**(8), **syslogd**(8)

NAME

hostrfs – convert IP addresses to RFS format

SYNOPSIS

hostrfs *hostname* [*portnum*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

hostrfs converts IP addresses to a format suitable for use by Remote File Sharing (RFS). It takes a hostname and an optional portnumber and produces an address in the following format:

> **\x<AF-INET>**<*portnum*><*IP-address*>**0000000000000000**

Each field given above is a hex ASCII representation. The **AF_INET** field is the address family which always has the value **0002**. *portnum* is the two-byte TCP port number; if not specified on the command line it defaults to **1450**. *IP-address* is the IP address of the *hostname* given on the command line followed by 16 trailing zeroes.

The output of this command may be directly used as the network address field for the address of an RFS name server in the **rfmaster**(5) file. It may also be used as input to the **nlsadmin** (8) command to initialize the addresses on which the **listener** program listens for service requests.

EXAMPLES

The output of

> **example% hostrfs wopr**

is

> **\00021450819035090000000000000000**

The output of the command can be used to initialize the network address on which the RFS **listener** program listens for remote service requests, for example:

> **example# nlsadmin –l 'hostrfs wopr' tcp**

SEE ALSO

rfmaster(5), **nlsadmin**(8)

System and Network Administration

NAME

htable – convert DoD Internet format host table

SYNOPSIS

/usr/etc/htable *filename*

DESCRIPTION

htable converts a host table in the format specified by RFC 952 to the format used by the network library routines. Three files are created as a result of running **htable**: **hosts**, **networks**, and **gateways**. The **hosts** file is used by the **gethostent**(3N) routines in mapping host names to addresses. The **networks** file is used by the **getnetent**(3N) routines in mapping network names to numbers. The **gateways** file is used by the routing daemon in identifying ''passive'' Internet gateways; see **routed**(8C) for an explanation.

If any of the files **localhosts**, **localnetworks**, or **localgateways** are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

htable is best used in conjunction with the **gettable**(8C) program which retrieves the DoD Internet host table from a host.

FILES

localhosts
localnetworks
localgateways

SEE ALSO

intro(3), **gethostent**(3N), **getnetent**(3N), **gettable**(8C), **routed**(8C)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *DoD Internet Host Table Specification*, RFC 952, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

BUGS

Does not properly calculate the **gateways** file.

NAME

icheck – file system storage consistency check

SYNOPSIS

/usr/etc/icheck [**–s**] [**–b** *numbers*] [*filesystem*]

DESCRIPTION

Note: **icheck** has been superseded for normal consistency checking by **fsck**(8).

icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. The normal output of **icheck** includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; that is, not in any file nor in the free list.

With the **–s** option **icheck** ignores the actual free list and reconstructs a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the superblock will not continue to be used. Notice also that the words in the superblock which indicate the size of the free list and of the i-list are believed. If the superblock has been curdled these words will have to be patched. The **–s** option suppresses the normal output reports.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

SEE ALSO

fs(5), **clri**(8), **dcheck**(8), **fsck**(8), **ncheck**(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) **icheck** announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and **icheck** considers it to contain 0.

Bad freeblock
> means that a block number outside the available space was encountered in the free list.

n **dups in free**
> means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since **icheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous superblocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

idload – RFS user and group mapping

SYNOPSIS

idload [**–n**] [**–g** *g_rules*] [**–u** *u_rules*] [*directory*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

idload is used on Remote File Sharing (RFS) servers to build translation tables for user and group IDs. It takes your **/etc/passwd** and **/etc/group** files and produces translation tables for user and group IDs from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you will need copies of remote **/etc/passwd** and **/etc/group** files. If no rules files are specified, remote user and group IDs are mapped to MAXUID+1. This is an ID number that is one higher than the highest number you could assign on your system.

By default, the remote password and group files are assumed to reside in **/usr/nserve/auth.info/***domain***/***host***/[passwd | group]**. The *directory* argument indicates that some directory structure other than **/usr/nserve/auth.info** contains the *domain/host* **passwd** and **group** files. *host* is the name of the host the files are from and *domain* is the domain where *host* can be found.

This command is restricted to the super-user.

This command is run automatically when the first remote mount is done of a remote resource (see **mount**(8)).

If any of the following are true, an error message will be sent to standard error.

- Neither rules files can be found or opened.
- There are syntax errors in the rules file.
- There are semantic errors in the rules file.
- Host information could not be found.
- The command is not run with super-user privileges.

Partial failures will display a warning message, although the process will continue.

OPTIONS

–n Do not produce a translation table, however, send a display of the ID mapping to the standard out. This is used to do a trial run of the mapping.

–u *u_rules* The *u_rules* file contains the rules for user ID translation. The default rules file is **/usr/nserve/auth.info/uid.rules.**

–g *g_rules* The *g_rules* file contains the rules for group ID translation. The default rules file is **/usr/nserve/auth.info/gid.rules.**

USAGE

Rules

The rules files have two types of sections, both optional: **global** and **host**. There can be only one global section, though there can be one host section for each host you want to map.

The **global** section describes the default conditions for translation for any machines that are not explicitly referenced in a **host** section. If the global section is missing, the default action is to map all remote user and group IDs from undefined hosts to MAXUID+1. The syntax of the first line of the **global** section is:

global

A **host** section is used for each client machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each **host** section is:

host*name*[. . .]

where *name* is replaced by the full name(s) of a host (*domain.hostname*).

The format of a rules file is described below. All lines are optional, but must appear in the order shown.

global
default *local* | **transparent**
exclude
[*remote_id–remote_id*] | [*remote_id*]
map [*remote_id*:*local*]

host *domain.hostname* [*domain.hostname* . . .]
default *local* | **transparent**
exclude [*remote_id–remote_id*] | [*remote_id*] | [*remote_name*]
map [*remote*:*local*] | *remote* | **all**

Each of these instruction types is described below.

The line

default *local* | **transparent**

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. **transparent** means that all remote user and group IDs will have the same numeric value locally unless they appear in the **exclude** instruction. *local* can be replaced by a local user name or ID to map all users into a particular local name or ID number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login ID .

The line

exclude [*remote_id–remote_id*] | [*remote_id*] | [*remote_name*]

defines remote IDs that will be excluded from the **default** mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of ID numbers, a single ID number, or a single name. (*remote_name* cannot be used in a global block.)

The line

map [*remote*:*local*] | *remote* | **all**

defines the local IDs and names that remote IDs and names will be mapped into. *remote* is either a remote ID number or remote name; *local* is either a local ID number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or ID will assign the user or group permissions of the same local name or ID. **all** is a predefined alias for the set of all user and group IDs found in the local **/etc/passwd** and **/etc/group** files. You cannot map by remote name in **global** blocks.

Note: **idload** will always output warning messages for '**map all**', since password files always contain multiple administrative user names with the same ID number. The first mapping attempt on the ID number will succeed, all subsequent attempts will fail.

RFS does not need to be running to use **idload**.

EXIT STATUS

On successful completion, **idload** will produce one or more translation tables and return a successful exit status. If **idload** fails, the command will return an unsuccessful exit status without producing a translation table.

FILES

/etc/passwd
/etc/group
/usr/nserve/auth.info/*domain***/***host***/[user | group]**
/usr/nserve/auth.info/vid.rules
/usr/nserve/auth.info/gid.rules

SEE ALSO

mount(8)

NAME

ifconfig – configure network interface parameters

SYNOPSIS

/usr/etc/ifconfig *interface* [*address_family*] [*address* [*dest_address*]] [**netmask** *mask*]
[**broadcast** *address*] [**up**] [**down**] [**trailers**] [**–trailers**] [**arp**] [**–arp**] [**private**]
[**–private**] [**metric** *n*]

/usr/etc/ifconfig *interface* [*protocol_family*]

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or to configure network interface parameters. **ifconfig** must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. Used without options, **ifconfig** displays the current configuration for a network interface. If a protocol family is specified, **ifconfig** will report only the details specific to that protocol family. Only the super-user may modify the configuration of a network interface.

The *interface* parameter is a string of the form *nameunit*, for example **ie0**. The interface name ''*–a*'' is reserved, and causes the remainder of the arguments to be applied to each address of each interface in turn.

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, the parameters and addresses are interpreted according to the rules of some address family, specified by the *address_family* parameter. The address families currently supported are **ether** and **inet**. If no address family is specified, **inet** is assumed.

For the TCP/IP family (**inet**), the address is either a host name present in the host name data base (see **hosts**(5)) or in the Network Interface Service (NIS) map **hosts**, or a TCP/IP address expressed in the Internet standard "dot notation". Typically, an Internet address specified in dot notation will consist of your system's network number and the machine's unique host number. A typical Internet address is **192.9.200.44**, where **192.9.200** is the network number and **44** is the machine's host number.

For the **ether** address family, the address is an Ethernet address represented as *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff. Only the super-user may use the **ether** address family.

If the *dest_address* parameter is supplied in addition to the *address* parameter, it specifies the address of the correspondent on the other end of a point to point link.

OPTIONS

up Mark an interface "up". This happens automatically when setting the first address on an interface. The **up** option enables an interface after an **ifconfig down**, reinitializing the hardware.

down Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

trailers This flag used to cause a non-standard encapsulation of inet packets on certain link levels. Sun drivers no longer use this flag, but it is ignored for compatibility.

–trailers Disable the use of a "trailer" link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between TCP/IP addresses and 10Mb/s Ethernet addresses.

–arp Disable the use of the Address Resolution Protocol.

private Tells the **in.routed** routing daemon (see **routed**(8C)) that the interface should not be advertised.

−private Specify unadvertised interfaces.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol (**routed**(8C)). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

netmask mask (**inet** only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation address, or with a pseudo-network name listed in the network table **networks**(5). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion. If a '+' (plus sign) is given for the netmask value, then the network number is looked up in the NIS **netmasks.byaddr** map (or in the **/etc/netmasks**) file if not running the NIS service.

broadcast *address*

(**inet** only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 0's. A + (plus sign) given for the broadcast value causes the broadcast address to be reset to a default appropriate for the (possibly new) address and netmask. Note that the arguments of **ifconfig** are interpreted left to right, and therefore

ifconfig −a netmask + broadcast +

and

ifconfig −a broadcast + netmask +

may result in different values being assigned for the interfaces' broadcast addresses.

EXAMPLES

If your workstation is not attached to an Ethernet, the **ie0** interface should be marked "down" as follows:

ifconfig ie0 down

To print out the addressing information for each interface, use

ifconfig −a

To reset each interface's broadcast address after the netmasks have been correctly set, use

ifconfig −a broadcast +

FILES

/dev/nit
/etc/netmasks

SEE ALSO

intro(3), **ethers**(3N), **arp**(4P), **hosts**(5), **netmasks**(5), **networks**(5) **netstat**(8C), **rc**(8), **routed**(8C).

DIAGNOSTICS

Messages indicating the specified interface does not exist, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

imemtest – stand alone memory test

SYNOPSIS

b /stand/imemtest

DESCRIPTION

imemtest runs stand alone, not under control of the operating system. With the PROM resident monitor in control of the system, type the command:

> **> b /stand/imemtest**

and the monitor boots the memory test program into memory. **imemtest** is completely self-explanatory. It prompts for all start and end addresses, and after that it runs under its own steam. It reports any errors it finds on the screen.

NAME

inetd – Internet services daemon

SYNOPSIS

/usr/etc/inetd [**–d**] [*configuration-file*]

DESCRIPTION

inetd, the Internet services daemon, is normally run at boot time by the **/etc/rc.local** script. When started **inetd** reads its configuration information from *configuration-file*, the default being **/etc/inetd.conf**. See **inetd.conf**(5) for more information on the format of this file. It listens for connections on the Internet addresses of the services that its configuration file specifies. When a connection is found, it invokes the server daemon specified by that configuration file for the service requested. Once a server is finished, **inetd** continues to listen on the socket (except in some cases which will be described below).

Depending on the value of the "wait-status" field in the configuration line for the service, **inetd** will either wait for the server to complete before continuing to listen on the socket, or immediately continue to listen on the socket. If the server is a "single-threaded" datagram server (a "wait-status" field of "wait"), **inetd** must wait. That server will handle all datagrams on the socket. All other servers (stream and ×lti-threaded" data-gram, a "wait-status" field of "nowait") operate on separate sockets from the connection request socket, thus freeing the listening socket for new connection requests.

Rather than having several daemon processes with sparsely distributed requests each running concurrently, **inetd** reduces the load on the system by invoking Internet servers only as they are needed.

inetd itself provides a number of simple TCP-based services. These include **echo**, **discard**, **chargen** (character generator), **daytime** (human readable time), and **time** (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). For details of these services, consult the appropriate RFC, as listed below, from the Network Information Center.

inetd rereads its configuration file whenever it receives a hangup signal, **SIGHUP**. New services can be activated, and existing services deleted or modified in between whenever the file is reread.

SEE ALSO

inetd.conf(5), **comsat**(8C), **ftpd**(8C), **rexecd**(8C), **rlogind**(8C), **rshd**(8C), **telnetd**(8C), **tftpd**(8C)

Postel, Jon, *Echo Protocol*, RFC 862, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, *Discard Protocol*, RFC 863, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, *Character Generator Protocol*, RFC 864, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, *Daytime Protocol*, RFC 867, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, and Ken Harrenstien, *Time Protocol*, RFC 868, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

infocmp [**–cdnILCruvV1**] [**–sd**] [**–si**] [**–sl**] [**–sc**] [**–w** *width*] [**–A** *directory*] [**–B** *directory*]
[*termname* ...]

SYNOPSIS

/usr/5bin/infocmp *arguments*

Note: *arguments* to **/usr/5bin/infocmp** are the same as those for **infocmp**, above.

AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

infocmp compares a binary **terminfo**(5V) entry with other terminfo entries, rewrites a **terminfo** description to take advantage of the **use=** *field*, or prints out a **terminfo** description from the corresponding binary file in a variety of formats. It displays boolean fields first, then numeric fields, then string fields.

It can also convert a **terminfo** entry to a **termcap**(5) entry; the –C flag causes **infocmp** to perform this conversion. Some **termcap** variables are not supported by **terminfo**, but those that can be derived from *terminfo* variables are displayed. Not all **terminfo** capabilities are translated either; only those that are allowed in a **termcap** entry are normally displayed. Specifying the **–r** option eliminates this restriction, allowing all capabilities to be displayed in **termcap** form.

Because padding is collected at the beginning of a capability, not all capabilities are displayed. Since mandatory padding is not supported by **terminfo** and **termcap** strings are not as flexible, it is not always possible to convert a **terminfo** string capability into an equivalent working **termcap** capability. Also, a subsequent conversion of the **termcap** file back into **terminfo** format will not necessarily reproduce the original source; **infocmp** attempts to convert parameterized strings, and comments out those that it can not.

Some common **terminfo** parameter sequences, their **termcap** equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

If no *termname* arguments are given, the environment variable TERM is used for all expected *termname* arguments.

OPTIONS

Default Options

If no options are specified and either zero or one *termname* is specified, the –I option is assumed to be in effect. If more than one *termname* is specified, the **–d** option is assumed.

Comparison Options

infocmp compares the description of the first terminal *termname* with each of the descriptions for terminals listed in subsequent *termname* arguments. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, –1 for integer variables, and NULL for string variables.

–c Produce a list of capabilities common to both entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **–u** option is worth using.

−d Produce a list of capabilities that differ between descriptions.

−n Produce a list of capabilities in neither entry.

Source Listing Options

The **−I**, **−L**, and **−C** options produce a source listing for each terminal named.

−I Use the **terminfo** names.

−L Use the long C variable name listed in **<term.h>**.

−C Display only those capabilities that have **termcap** equivalents, using the **termcap** names and displaying them in **termcap** form whenever possible.

The source produced by the **−C** option may be used directly as a **termcap** entry, but not all of the parameterized strings may be changed to the **termcap** format. All padding information for strings is collected together and placed at the beginning of the string where **termcap** expects it. Mandatory padding (padding information with a trailing '/') will become optional.

−r When using **−C**, display all capabilities, not just those capabilities that have **termcap** equivalents.

−u Produce a **terminfo** source description for the first named terminal which is relative to the descriptions given by the entries for all terminals named subsequently on the command line, by analyzing the differences between them, and producing a description with **use**= fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using **infocmp** will show what can be done to change one description to be relative to the other.

A capability is displayed with an at-sign (@) if it no longer exists in the first terminal, but one of the other terminal entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries has a different value for that capability.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(8V) does a left-to-right scan of the capabilities, specifying two **use**= entries that contain differing entries for the same capabilities will produce different results, depending on the order in which they are given. **infocmp** flags any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability after a **use**= entry that contains it, will cause the second specification to be ignored. Using **infocmp** to recreate a description can be a useful check to make sure that everything was specified correctly in the original.

Specifying superfluous **use**= slows down the comparison, but is not fatal; **infocmp** flags superfluous **use**= fields.

Sorting Options

−sd Sort fields in the order that they are stored in the **terminfo** database.

−si Sort fields by **terminfo** name.

−sl Sort fields by the long C variable name.

−sc Sort fields by the **termcap** name.

If no sorting option is given, fields are sorted alphabetically by the **terminfo** name within each type, except in the case of the **−C** or the **−L** options, which cause the sorting to be done by the **termcap** name or the long C variable name, respectively.

Changing Databases

The location of the compiled **terminfo** database is taken from the environment variable TERMINFO. If the variable is not defined, or if the terminal is not found in that location, the system **terminfo** database, usually in **/usr/share/lib/terminfo**, is used. The options **−A** and **−B** may be used to override this location. With these options, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people.

−A Set TERMINFO for the first *termname* argument.

−B Set TERMINFO for the remaining *termname* arguments.

Other Options

−v Print out tracing information on the standard error.

−V Print out the version of the program in use on the standard error and exit.

−1 Print fields out one to a line. Otherwise, fields are printed several to a line to a maximum width of 60 characters.

−w *width*
Change the output to *width* characters.

FILES

/usr/share/lib/terminfo/?/*
compiled terminal description database

/usr/5include/term.h

SEE ALSO

curses(3V), **termcap**(5), **terminfo**(5V), **tic**(8V)

DIAGNOSTICS

malloc is out of space!
There was not enough memory available to process all the terminal descriptions requested. Run **infocmp** in several smaller stages (with fewer *termname* arguments).

use= order dependency found:
A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.
A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparision to be done.
The **−u**, **−d** and **−c** options require at least two terminal names.

NAME

init – process control initialization

SYNOPSIS

/usr/etc/init [**–bs**]

DESCRIPTION

init is invoked inside the operating system as the last step in the boot procedure. It normally runs the sequence of commands in the script **/etc/rc.boot** (see **rc**(8)) to check the file system. If passed the **–b** option from the boot program, **init** skips this step. If the file system check succeeds or is skipped, **init** runs the commands in **/etc/rc** and **/etc/rc.local** to begin multiuser operation; otherwise it commences single-user operation by giving the super-user a shell on the console. It is possible to pass the **–s** parameter from the boot program to **init** so that single-user operation is commenced immediately.

Whenever a single-user shell is created, and the system is running as a secure system, the **init** program demands the super-user password. This is to prevent an ordinary user from invoking a single-user shell and thereby circumventing the system's security. Logging out (for instance, by entering an EOT) causes **init** to proceed with a multi-user boot. The super-user password is demanded whenever the system is running secure as determined by **issecure**(3), or the console terminal is not labeled "secure" in **/etc/ttytab**.

Whenever single-user operation is terminated (for instance by killing the single-user shell) **init** runs the scripts mentioned above.

In multi-user operation, **init**'s role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file **/etc/ttytab** and executes a command for each terminal specified in the file. This command will usually be **/usr/etc/getty**. **getty**(8) opens and initializes the terminal line, reads the user's name and invokes **login**(1) to log in the user and execute the shell.

Ultimately the shell will terminate because it received EOF, either explicitly, as a result of hanging up, or from the user logging out. The main path of **init**, which has been waiting for such an event, wakes up and removes the appropriate entry from the file **/etc/utmp**, which records current users. **init** then makes an entry in **/var/adm/wtmp**, which maintains a history of logins and logouts. The **/var/adm/wtmp** entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and the command for that terminal is reinvoked.

init catches the *hangup* signal (SIGHUP) and interprets it to mean that the file **/etc/ttytab** should be read again. The shell process on each line which used to be active in **/etc/ttytab** but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add terminal lines without rebooting the system by changing **/etc/ttytab** and sending a *hangup* signal to the **init** process: use '**kill –HUP 1**'.

init terminates multi-user operations and resumes single-user mode if sent a terminate (SIGTERM) signal: use '**kill –TERM 1**'. If there are processes outstanding which are deadlocked (due to hardware or software failure), **init** does not wait for them all to die (which might take forever), but times out after 30 seconds and prints a warning message.

init ceases to create new processes, and allows the system to slowly die away, when sent a terminal stop (SIGTSTP) signal: use '**kill –TSTP 1**'. A later hangup will resume full multi-user operations, or a terminate will initiate a single-user shell. This hook is used by **reboot**(8) and **halt**(8).

Whenever it reads **/etc/ttytab**, **init** will normally write out an old-style **/etc/ttys** file reflecting the contents of **/etc/ttytab**. This is required in order that programs built on earlier versions of SunOS that read the **/etc/ttys** file (for example, programs using the **ttyslot**(3V) routine, such as **shelltool (1)**) may continue to run. If it is not required that such programs run, **/etc/ttys** may be made a link (hard or symbolic) to **/etc/ttytab** and **init** will not write to **/etc/ttys**.

init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the **init** program cannot be located, the system will print an error message and panic.

FILES

/dev/console
/dev/tty*
/etc/utmp
/var/adm/wtmp
/etc/ttytab
/etc/rc
/etc/rc.local
/etc/rc.boot
/usr/etc/getty

SEE ALSO

kill(1), **login**(1), **sh**(1), **shelltool**(1), **issecure**(3), **ttyslot**(3V), **ttytab**(5), **getty**(8), **halt**(8), **rc**(8), **reboot**(8), **shutdown**(8)

DIAGNOSTICS

command **failing, sleeping.**

A process being started to service a line is exiting quickly each time it is started. This is often caused by a ringing or noisy terminal line. **init** will sleep for 30 seconds, then continue trying to start the process.

WARNING: Something is hung (won't die); ps axl advised.

A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

NAME

installboot – install bootblocks in a disk partition

SYNOPSIS

/usr/mdec/installboot [**–lvt**] *bootfile protobootblk bootdevice*

DESCRIPTION

The **boot**(8S) program is loaded from disk by bootblock code which resides in the bootblock area of a disk partition. In order for the bootblock code to read the boot program (usually **/boot**) it is necessary for it to know the block numbers occupied by the boot program. Previous versions of the bootblock code could find **/boot** by interpreting the file system on the partition from which it was being booted, but this is no longer so.

installboot plugs the block numbers of the boot program into a table in the bootblock code, and writes the modified bootblock code onto the disk. Note: **installboot** must be run every time the boot program is reinstalled, since in general, the block list of the boot program will change each time it is written.

bootfile is the name of the boot program, usually **/boot**. *protobootblk* is the name of the bootblock code into which the block numbers of the boot program are to be inserted. The file read in must have an **a.out**(5) header, but it will be written out to the device with the header removed. *bootdevice* is the name of the disk device onto which the bootblock code is to be installed.

OPTIONS

–l Print out the list of block numbers of the boot program.

–t Test. Display various internal test messages.

–v Verbose. Display detailed information about the size of the boot program, etc.

EXAMPLE

To install the bootblocks onto the root partition on a Xylogics disk:

```
example% cd /usr/mdec
example% installboot –vlt /boot bootxy /dev/rxy0a
```

For an SD disk, you would use **bootsd** and **/dev/rsd0a**, respectively, in place of **bootxy** and **/dev/rxy0a**.

SEE ALSO

od(1V), **a.out**(5) **boot**(8S), **bootparamd**(8), **init**(8), **kadb**(8S), **monitor**(8S), **ndbootd**(8C), **rc**(8), **reboot**(8)

System and Network Administration

Installing SunOS 4.1

NAME

install_small_kernel – install a small, pre-configured kernel

SYNOPSIS

/usr/etc/install/install_small_kernel [*hostname*] ...

DESCRIPTION

install_small_kernel is a script that installs a small, pre-configured kernel, **GENERIC_SMALL** on a host. This kernel supports approximately four users, and is only available for the following configurations:

Sun-3/50 and Sun-3/60 systems with up to 2 SCSI disks, 1 SCSI tape
Sun-3/80 systems with up to 4 SCSI disks, 1 SCSI tape
Sun-4/110 systems with up to 2 SCSI disks, 1 SCSI tape
SPARCsystem 330 systems with up to 4 SCSI disks, 1 SCSI tape
SPARCstation 1 systems with up to 4 SCSI disks. 1 floppy drive and 2 SCSI tapes

If *hostname* is a server that does not fit any of the above configurations, **install_small_kernel** can be used to install the small kernel on its clients.

If no hostnames are specified, **install_small_kernel** cycles through all the clients configured for a server to determine the small kernel installs to be made. If the '**small_kernel**' flag in the client file, **/etc/install/client.***hostname* is set to '**yes**', that client will not be processed. To force re-installation of a small kernel on any clients, simply call **install_small_kernel** with the appropriate client names.

install_small_kernel prompts for confirmation before actually doing the install on any host.

install_small_kernel is executable from the miniroot, as well as single-user and multi-user modes. It supports standalone and server configuration in all cases, but dataless systems are supported in multi-user mode only. This script is restricted to the super-user.

FILES

/usr/sys/sun*arch***/conf/GENERIC_SMALL**
kernel configuration file for *arch* **/usr/install/client.***hostname*

SEE ALSO

add_client(8), **add_services**(8), **rm_client**(8), **suninstall**(8)

System and Network Administration

NAME

installtxt, gencat – create a message archive

SYNOPSIS

/usr/etc/installtxt [[–]**d** | **c** | **r** | **t** | **x** | **i** [**ouvs**]]] *message-archive*... [*source-message-file*]

/usr/etc/gencat *catfile msgfile*...

DESCRIPTION

installtxt converts each *source-message-file* into a binary format message archive. At the same time, if necessary, **installtxt** maintains groups of files (member files) combined into a single message archive. **installtxt** is normally used to create and update message archives used by the run-time message handling facility **gettext**(3).

gencat performs the same function as **installtxt**, but supports the X/Open catalog source format.

installtxt creates the message archive in *message-archive*. If the message archive does not exist, it is created by the **–c** option. *source-message-file* contains source versions of the target strings. On successful completion of an update operation of **installtxt,** the message archive will have been updated with details of the formatted version of each *source-message-file*. If *message-archive* does not contain the full pathname of the run-time location of the message catalog, it will have to be moved to the appropriate locale directory before applications using the archive are activated.

gencat merges the message text source files (*msgfile*...) into a formatted message catalog *catfile*. *catfile* is created if it does not already exist. If *catfile* does exist, its messages are included in the new *catfile*. If set and message numbers collide, the new message-text defined in *msgfile* will replace the old message text currently contained in *catfile*. The output formats of both *message_archive* and *catfile* are the same. However it should be noted that on a per-application basis, it is not intended that the output forms of these two utilities should be mixed, and the consequence of doing so is undefined.

OPTIONS

The following options and modifiers apply to **installtxt** only. For **installtxt** you must indicate only one of: **c, d, r, t,** or, **x**, which may be followed by one or more **Modifiers, o, u,** or , **v**.

The options are:

c Create. The member file called *source-message-file* is being made for the first time in the message archive. It should not exist already.

d Delete the named member files from *message archive*. Note that individual messages can be deleted by entering an empty value after the message-id selecting the message to be deleted. With the **v** option these deletions are notified on the standard output.

r Replace the named member files in the message archive. This allows the existing *message archive* to be merged with new versions of messages. No new message will be added to the message archive unless each message-tag in the *source-message-file* is unique in the active domain. If the member file contains a message-tag that is not unique within the active domain, **installtxt** will fail and the contents of the active message archive will not be altered.

t Table of contents. Produces a list on the standard output of all member files in *message_archive*.

x Extract. If no names are given, all member files in the message archive are extracted into the current directory; if names are given, only those files are extracted. In neither case does **x** alter the message archive. The extracted member files will be returned in their original source format. It is possible for the **–x** option to lose comments that were contained in the original source message file. In addition, overlong lines may be escaped (using \n) at a point that is different from the original source, although the end result will logically be the same string.

Modifiers

o Old date. When member files are extracted with the **x** option, set the "last modified" date to the date recorded in the message archive.

u Update. Replace only those member files that have changed since they were put in the message archive. Used with the **r** option.

v Verbose. When used with the **c**, **r**, or **d** option, give a file-by-file description of the creation of a new *message archive* file from the old version and the constituent member files. When used with **x**, give a file-by-file description of the extraction of message archive member files. When used with **t**, print information about the size and creation date of the message archive, as well as a count of the number of target strings in the message-archive.

USAGE

source-message-file consists of one or more lines of text, with each line containing either a comment, a directive or a text line. The format of a comment line is:

"$ %s", *comment*

A line beginning with a dollar sign ($), followed by a *blank* character streated as a comment line. The format of directives is:

"$%s %s", *control-type*, *value*

Directives should be directly preceded by a dollar sign ($), and followed by an optional value. There is one *blank* character between the directive and its value. The following directives are recognized:

$separator *c*

This directive specifies an optional separator character that will subsequently be used in the following text lines to separate the message identifier from the target string. There is one *blank* character between **separator** and the separator character itself. If this line is absent then the default separator is the *blank* character. Only the first occurrence of this character on one text line will be interpreted, for example:

$separator :
12345:Bonjour: Mon ami

would declare the message identifier to be 12345, the target string would contain the second ":".

$domain *domain*

This directive states that all following target strings are contained within a domain of the object message file as described by *domain*. *domain* can be any string of up to {PATH_MAX} bytes in length.

$quote *c* This directive specifies an optional quote character *c*, which can be used to surround both *message_string* and *message_identifier* . By default, or if an empty **$quote** directive is supplied, no quoting of *message_string* will be recognized. If the **$quote** directive is given then all message strings must contain pairs of quotes, although quotes around the message_identifier are still optional after the directive.

The format of the text line is:

"%s%s%s", *message_identifier*, *separator_character*, *message_string*

Each line defines a message identifier and a target string pair.

Empty lines in a source text file are ignored. If a *message_identifier* starts with a dollar ($) character, then that dollar character must be escaped with a backslash (\$). Any other form of input line syntax is illegal and will cause **installtxt** to exit with the error value.

Message strings and message identifiers can contain the special characters and escape sequences as defined in the following table:

Description	**Symbol**
newline	\n
tab	\t
vertical-tab	\v
backspace	\b
carriage-return	\r
form-feed	\f
backslash	\\
bit pattern	**\ddd**

The escape sequence *\ddd* consists of backslash followed by 1, 2 or 3 octal digits, which are used to specify the value of the desired character. If *message_identifier* contains the separator character then it must be escaped with a backslash (\) character. If the character following a backslash is not one of those specified, the effect is unspecified.

Backslash, \, followed by a NEWLINE character is used to continue an individual string on the following line. Both *message_identifier* and *message_string* may be continued over lines in this way. *message_string* is stored in *object_file* in an implementation specific way. If *message_string* is empty, and *separator* is present, a null string is stored in *object_file*.

msgfile must be in the X/Open **gencat** format.

EXAMPLES

```
# /bin/sh script
# The following creates a message archive in the file messages.general
installtxt –cv messages.general input
#
```

FILES

/etc/locale/LC_MESSAGES/*locale***/***domain*
: standard private location for message archive/catalog in locale *locale* and domain *domain*

/usr/share/lib/locale/LC_MESSAGES
: standard shared location for message archive/catalog in locale *locale* and domain *domain*

SEE ALSO

catgets(3), **gettext**(3), **setlocale**(3V), **locale**(5)

X/Open Portability Guide Issue 2

NAME

intr – allow a command to be interruptible

SYNOPSIS

intr [**–anv**] [**–t** *seconds*] *command* [*arguments*]

DESCRIPTION

intr executes *command* after altering the execution environment to make *command* to be interrutable.

Since interactive commands are by default interruptable, **intr** is intended for use as a wrapper around commands started by the **/etc/rc** files; commands spawned from these files are not interruptable by default. It has no other intended use than as a wrapper around **/etc/rc** commands.

The following signals are ignored as a result of wrapping **intr** around a command:

SIGTSTP	terminal generated stop signal
SIGTTIN	background read
SIGTTOU	background write

The following signals are reset to their default actions:

SIGINT	interrupt signal
SIGQUIT	quit signal

OPTIONS

–v Echo the command in the form ' *command*' (note leading SPACE).

–a Echo the command and its arguments.

–n Do not echo a NEWLINE after the command or arguments (for example '**echo –n ...**').

–t *secs* Arrange to have a **SIGALRM** signal delivered to the command in *secs* seconds.

EXAMPLES

All of these examples assume that they are in an **/etc/rc** file, that is, talking to the console, and not run interactively. The following example runs **fsck**(8) but allow it to be killed from the console:

intr fsck –p –w / /usr

Echoing is provided so that

ypbind; echo –n ' ypbind'

can be replaced with

intr –vn ypbind

Timeouts are provided so that the machine will not hang at boot:

intr –t 10 rdate *date_host*

SEE ALSO

echo(1V), **login**(1), **init**(8), **rc**(8)

BUGS

The –v option is a kludge.

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [**–cdDIt**] [**–l** *n*] [*disk* ...] [*interval* [*count*]]

DESCRIPTION

iostat can iteratively report terminal and disk I/O activity, as well as CPU utilization. The first report is for all time since a reboot and each subsequent report is for the prior interval only.

In order to compute this information, the kernel maintains a number of counters. For each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, at each clock tick, the state of each disk is examined and a tally is made if the disk is active. The kernel also provides approximate transfer rates of the devices.

OPTIONS

iostat's activity class options default to **tdc** (terminal, disk, and CPU). If any activity class options are specified, the default is completely overridden. Therefore, if only **–d** is specified, neither terminal nor CPU statistics will be reported. The last disk option specified (either **–d** or **–D**) is the only one that is used.

–c Report the percentage of time the system has spent in user mode, in user mode running low priority processes, see **nice**(1), in system mode, and idling.

–d For each disk, report the number of kilobytes transferred per second, the number of transfers per second, and the milliseconds per average seek (see BUGS below).

–D For each disk, report the reads per second, writes per second, and percentage disk utilization.

–I Report the counts in each interval, rather than reporting rates.

–t Report the number of characters read and written to terminals.

–l *n* Limit the number of disks included in the report to *n*; the disk limit defaults to 4. Note: disks explicitly requested (see *disk* below) are not subject to this disk limit.

disk Explicitly specify the disks to be reported; in addition to any explicit disks, any active disks up to the disk limit (see **–l** above) will also be reported.

interval Report once each *interval* seconds.

count Only print *count* reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(8)

BUGS

Milliseconds per average seek is an approximation based on the disk (not the controller) transfer rate. Therefore, the seek time will be over-estimated in systems with slower controllers.

NAME

ipallocd – Ethernet-to-IP address allocator

SYNOPSIS

/usr/etc/rpc.ipallocd

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ipallocd is a daemon that determines or temporarily allocates IP addresses within a network segment. The service is only available on the system which is home to the address authority for the network segment, currently the Network Interface Service (NIS) master of the **hosts.byaddr** map although the service is not tied to the NIS service. It has complete knowledge of the hosts listed in the NIS service, and, if the system is running the name server, of any hosts listed in internet domain tables automatically accessed on that host through the standard library **gethostent**(3N) call.

This protocol uses DES authentication (the Sun Secure RPC protocol) to restrict access to this function. The only clients privileged to allocate addresses are those whose net IDs are in the networks group. For machine IDs, the machine must be an NIS server.

The daemon uses permanent entries in the **/etc/ethers** and **/etc/hosts** files when they exist and are usable. In other cases, such as when a system is new to the network, **ipallocd** enters a temporary mapping in a local cache. Entries in the cache are removed when there have been no references to a given entry in the last hour. This cache survives system crashes so that IP addresses remain consistent.

The daemon also provides corresponding IP address to name mapping.

If the file **/etc/ipalloc.netrange** exists, **ipallocd** refuses to allocate addresses on networks not listed in the **netrange** file, or for which no free address is available.

FILES

/etc/ipalloc.cache	temporary cache
/etc/ipalloc.netrange	optional file to allocate network addresses

SEE ALSO

ipalloc(3R), **pnp**(3R), **ipalloc.netrange**(5), **ipallocd**(8C), **netconfig**(8C), **pnpboot**(8C), **rarpd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

kadb – adb-like kernel and standalone-program debugger

SYNOPSIS

> b kadb [**–d**] [*boot-flags*]

DESCRIPTION

kadb is an interactive debugger that is similar in operation to **adb**(1), and runs as a standalone program under the PROM monitor. You can use **kadb** to debug the kernel, or to debug any standalone program.

Unlike **adb**, **kadb** runs in the same supervisor virtual address space as the program being debugged — although it maintains a separate context. The debugger runs as a *coprocess* that cannot be killed (no '**:k**') or rerun (no '**:r**'). There is no signal control (no '**:i**', '**:t**', or '**$i**'), although the keyboard facilities (CTRL–C, CTRL–S, and CTRL–Q) are simulated.

While the kernel is running under **kadb**, the abort sequence (**L1-A** or **BREAK**) drops the system into **kadb** for debugging — as will a system panic. When running other standalone programs under **kadb**, the abort sequence will pass control to the PROM monitor. **kadb** is then invoked from the monitor by jumping to the starting address for **kadb** found in **/usr/include/debug/debug.h** The following list gives the monitor commands to use for each system.

System	Monitor Command
Sun-2	**g fd00000**
Sun-3	**g fd00000**
Sun386i	**g fe005000**
Sun-4	**g ffc00000**
SPARCstation 1	**go ffc00000**

The **kadb** user interface is similar to that of **adb**. Note: **kadb** prompts with

kadb>

Most **adb** commands function in **kadb** as expected. Typing an abort sequence in response to the prompt returns you to the PROM monitor, from which you can examine control spaces that are not accessible within **adb** or **kadb**. The PROM monitor command **c** will return control to **kadb**. As with '**adb –k**', **$p** works when debugging kernels (by actually mapping in new user pages). The verbs **?** and / are equivalent in **kadb** , since there is only one address space in use.

OPTIONS

kadb is booted from the PROM monitor as a standalone program. If you omit the **–d** flag, **kadb** automatically loads and runs **vmunix** from the filesystem **kadb** was loaded from. The **kadb vmunix** variable can be patched to change the default program to be loaded.

–d Interactive startup. Prompts with

kadb:

for a file to be loaded. From here, you can enter a boot sequence line to load a standalone program. Boot flags entered in response to this prompt are included with those already set and passed to the program. If you type a RETURN only, **kadb** loads **vmunix** from the filesystem that **kadb** was loaded from.

boot-flags

You can specify boot flags as arguments when invoking **kadb.** Note: **kadb** always sets the **–d** (debug) boot flag, and passes it to the program being debugged.

USAGE

Refer to **adb** in *Debugging Tools*.

Kernel Macros

As with **adb**, kernel macros are supported. With **kadb**, however, the macros are compiled into the debugger itself, rather than being read in from the filesystem. The **kadb** command **$M** lists macros known to **kadb.**

Setting Breakpoints

Self-relocating programs such as the SunOS kernel need to be relocated before breakpoints can be used. To set the first breakpoint for such a program, start it with '**:s**'; **kadb** is then entered after the program is relocated (when the system initializes its interrupt vectors). Thereafter, '**:s**' single-steps as with **adb**. Otherwise, use '**:c**' to start up the program.

Sun386i System Commands

The Sun386i system version of **kadb** has the following additional commands. Note, for the general syntax of **adb** commands, see **adb**(1).

:i Read a byte (with the INB instruction) in from the port at *address*.

:o Send a byte (with the OUTB instruction) containing *count* out through the port at *address*.

:p Like **:b** in **adb**(1), but sets a breakpoint using the hardware debug register instead of the breakpoint instruction. The advantage of using **:p** is that when setting breakpoints with the debug register it is not necessary to have write access to the breakpoint location. Four (4) breakpoints can be set with the hardware debug registers.

$S Switch I/O from the console to the serial port or vice versa.

[Like **:e** in **adb**(1), but requires only one keystroke and no RETURN character.

] Like :s in **adb**(1), but requires only one keystroke and no RETURN character.

Automatic Rebooting with kadb

You can set up your workstation to automatically reboot **kadb** by patching the *vmunix* variable in **/boot** with the string **kadb**. (Refer to **adb** in *Debugging Tools* for details on how to patch executables.)

FILES

/vmunix
/boot
/kadb
/usr/include/debug/debug.h

SEE ALSO

adb(1), **boot**(8S)

Debugging Tools
Writing Device Drivers

BUGS

There is no floating-point support, except on Sun386i systems.

kadb cannot reliably single-step over instructions that change the status register.

When sharing the keyboard with the operating system the monitor's input routines can leave the keyboard in a confused state. If this should happen, disconnect the keyboard momentarily and then reconnect it. This forces the keyboard to reset as well as initiating an abort sequence.

Most of the bugs listed in **adb**(1) also apply to **kadb**.

NAME

keyenvoy – talk to keyserver

SYNOPSIS

keyenvoy

DESCRIPTION

keyenvoy is used by some RPC programs to talk to the key server, **keyserv**(8C). The key server will not talk to anything but a root process, and **keyenvoy** is a set-uid root process that acts as an intermediary between a user process that wishes to talk to the key server and the key server itself.

This program cannot be run interactively.

SEE ALSO

keyserv(8C)

NAME

keyserv – server for storing public and private keys

SYNOPSIS

keyserv [**–dkn**]

DESCRIPTION

keyserv is a daemon that is used for storing the private encryption keys of each user logged into the system. These encryption keys are used for accessing secure network services such as secure NFS. When a user logs in to the system, the **login**(1) program uses the login password to decrypt the user's encryption key stored in the Network Interface Service (NIS), and then gives the decrypted key to the **keyserv** daemon to store away.

Normally, root's key is read from the file **/etc/.rootkey** when the daemon starts up. This is useful during power-failure reboots when no one is around to type a password, yet you still want the secure network services to operate normally.

OPTIONS

–d Prohibit the use of the default key. If this is used then every machine and user should have a publickey. New publickeys cannot be created if you do not already have a key. This can be done globally for an entire domain by deleting the **nobody** entry from **/etc/publickey** on the NIS master. See **chkey**(1)

–k Remember keylogins across machine reboots. This is only needed if **at**(1) is used to schedule jobs that require secure RPC. Use of this option is not recommended.

–n Do not read root's key from **/etc/.rootkey.** Instead, prompt the user for the password to decrypt **root 's** key stored in the NIS service and then store the decrypted key in **/etc/.rootkey** for future use. This option is useful if the **/etc/.rootkey** file ever gets out of date or corrupted.

FILES

/etc/.rootkey **/etc/keystore**

SEE ALSO

login(1), **keylogin**(1), **keylogout**(1), **publickey**(5)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

/usr/etc/kgmon [**–bhpr**] [*filesystem*] [*memory*]

DESCRIPTION

kgmon is a tool used when profiling the operating system. When no arguments are supplied, **kgmon** indicates the state of operating system profiling as running, off, or not configured (see **config**(8)). If the **–p** flag is specified, **kgmon** extracts profile data from the operating system and produces a **gmon.out** file suitable for later analysis by **gprof**(1).

OPTIONS

–b Resume the collection of profile data.

–h Stop the collection of profile data.

–p Dump the contents of the profile buffers into a **gmon.out** file.

–r Reset all the profile buffers. If the **–p** flag is also specified, the **gmon.out** file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling is momentarily suspended, the operating system profile buffers are dumped, and profiling is immediately resumed.

FILES

/vmunix	the default system
/dev/kmem	the default memory
gmon.out	

SEE ALSO

gprof(1), **config**(8)

DIAGNOSTICS

Users with only read permission on **/dev/kmem** cannot change the state of profiling collection. They can get a **gmon.out** file with the warning that the data may be inconsistent if profiling is in progress.

NAME

ldconfig – link-editor configuration

SYNOPSIS

/usr/etc/ldconfig [*directory* ...]

DESCRIPTION

ldconfig is used to configure a performance-enhancing cache for the run-time link-editor, **ld.so**. It is run from **/etc/rc.local** and periodically via **cron** to avoid linking with stale libraries. It should be also be run manually when a new shared object (e.g., a shared library) is installed on the system.

When invoked with no arguments, a default set of directories are built into the cache – these are the directories searched by default by the link editors. Additional directories may be specified on the command line.

FILES

/etc/ld.so.cache holds the cached data.

SEE ALSO

ld(1)

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

/usr/etc/link *filename1 filename2*

/usr/etc/unlink *filename*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

link and **unlink** perform their respective system calls on their arguments, abandoning all error checking.

SEE ALSO

rm(1), **link**(2V), **unlink**(2V)

WARNINGS

Only the super-user can unlink a directory, in which case the files it contains are lost. The files can, however, be recovered from the file system's **lost+found** directory after performing an **fsck**.

If you have write permission on the directory in which *filename* resides, **unlink** removes that file without warning, regardless of its ownership.

NAME

lockd, rpc.lockd – network lock daemon

SYNOPSIS

/usr/etc/rpc.lockd [**–g** *graceperiod*] [**–t** *timeout*]

DESCRIPTION

lockd processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. **lockd** forwards lock requests for remote data to the server site's lock daemon through the **rpc**(3N) **xdr**(3N) in **lockd**(8C) package. **lockd** then requests the status monitor daemon, **statd**(8C), for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client site lock daemons to submit reclaim requests. Client site lock daemons, on the other hand, are notified by the status daemon of the server recovery and promptly resubmit previously granted lock requests. If **lockd** fails to secure a previously granted lock at the server site, it sends SIGLOST to a process.

OPTIONS

–t *timeout* — Use *timeout* (seconds) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

–g *graceperiod* — Use *graceperiod* (**seconds**) as the grace period duration instead of the default value (45 seconds).

SEE ALSO

fcntl(2V), **lockf**(3), **signal**(3V), **statd**(8C)

NAME

logintool – graphic login interface

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

logintool is started by **getty**(8) to display a full screen window for logging in. It cannot be run from the shell. It is more attractive than the traditional '**login:** ' prompt, and also provides help for the person without a username and information about the workstation.

logintool is normally invoked on the console by **getty**(8), and works only on a frame buffer.

If the **newlogin** policy in the **policies** Network Interface Service (NIS) map is set to **unrestricted**, then **logintool** may create new user accounts in the NIS service. The account resides on the local system if it is diskful, or on the system's boot server if the local system is diskless.

FILES

/usr/share/lib/ez/login

SEE ALSO

getty(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

lpc – line printer control program

SYNOPSIS

/usr/etc/lpc [*command* [*parameter*...]]

DESCRIPTION

lpc controls the operation of the printer, or of multiple printers, as described in the **/etc/printcap** database. **lpc** commands can be used to start or stop a printer, disable or enable a printer's spooling queue, rearrange the order of jobs in a queue, or display the status of each printer—along with its spooling queue and printer daemon.

With no arguments, **lpc** runs interactively, prompting with **lpc>**. If arguments are supplied, **lpc** interprets the first as a *command* to execute; each subsequent argument is taken as a *parameter* for that command. The standard input can be redirected so that **lpc** reads commands from a file.

USAGE

Commands

Commands may be abbreviated to an unambiguous substring. Note: the *printer* parameter is specified just by the name of the printer (as **lw**), not as you would specify it to **lpr**(1) or **lpq**(1) (not as **–Plw**).

? [*command*]...

help [*command*]...

Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort [**all** | [*printer* ...]]

Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by **lpr**(1)) for the specified printers. The **abort** command can only be used by the super-user.

clean [**all** | [*printer* ...]]

Remove all files with names beginning with **cf**, **tf**, or **df** from the specified printer queue(s) on the local machine. The **clean** command can only be used by the super-user.

disable [**all** | [*printer* ...]]

Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by **lpr**(1). The **disable** command can only be used by the super-user.

down [**all** | [*printer* ...]] [*message*]

Turn the specified printer queue off, disable printing and put *message* in the printer status file. The message doesn't need to be quoted, the remaining arguments are treated like **echo**(1V). This is normally used to take a printer down and let others know why (**lpq**(1) indicates that the printer is down, as does the **status** command).

enable [**all** | [*printer* ...]]

Enable spooling on the local queue for the listed printers, so that **lpr**(1) can put new jobs in the spool queue. The **enable** command can only be used by the super-user.

exit

quit Exit from **lpc**.

restart [**all** | [*printer* ...]]

Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. **lpq**(1) reports that there is no daemon present when this condition occurs. This command can be run by any user.

start [**all** | [*printer* ...]]

Enable printing and start a spooling daemon for the listed printers. The **start** command can only be used by the super-user.

status [**all** | [*printer* ...]]
Display the status of daemons and queues on the local machine. This command can be run by any user.

stop [**all** | [*printer* ...]]
Stop a spooling daemon after the current job completes and disable printing. The **stop** command can only be used by the super-user.

topq *printer* [*job#* ...] [*user* ...]
Move the print job(s) specified by *job#* or those job(s) belonging to *user* to the top (head) of the printer queue. The **topq** command can only be used by the super-user.

up [**all** | [*printer* ...]] Enable everything and start a new printer daemon. Undoes the effects of **down**.

FILES

/etc/printcap	printer description file
/var/spool/*	spool directories
/var/spool/*/lock	lock file for queue control

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **printcap**(5), **lpd**(8)

DIAGNOSTICS

?Ambiguous command
The abbreviation you typed matches more than one command.

?Invalid command
You typed a command or abbreviation that was not recognized.

?Privileged command
You used a command can be executed only by the super-user.

NAME

lpd – printer daemon

SYNOPSIS

/usr/lib/lpd [**–l**] [**–L** *logfile*] [*port#*]

DESCRIPTION

lpd is the line printer daemon (spool area handler). It is usually invoked at boot time from the **rc**(8) script, making a single pass through the **printcap**(5) file to find out about the existing printers and printing any files left after a crash. It then accepts requests to print files in a queue, transfer files to a spooling area, display a queue's status, or remove jobs from a queue. In each case, it forks a child process for each request, and continues to listen for subsequent requests.

The Internet port number used to communicate with other processes is usually obtained with **getservent**(3N), but can be specified with the *port#* argument.

If a file cannot be opened, an error message is logged using the **LOG_LPR** facility of **syslog**(3). **lpd** will try up to 20 times to reopen a file it expects to be there, after which it proceeds to the next file or job.

OPTIONS

–l Log valid requests received from the network. This can be useful for debugging purposes.

–L *logfile*
Change the file used for writing error conditions to *logfile*. The default is to report a message using the **syslog**(3) facility.

OPERATION

Access Control

Access control is provided by two means. First, all requests must come from one of the machines listed in either the file **/etc/hosts.equiv** or **/etc/hosts.lpd**. (This latter file is in **hosts.equiv**(5) format.) Second, if the **rs** capability is specified in the **printcap** entry, **lpr**(1) requests are only be honored for users with accounts on the printer host.

Lock File

The **lock** file in each spool directory is used to prevent multiple daemons from becoming active, and to store information about the daemon process for **lpr**(1), **lpq**(1), and **lprm**(1).

lpd uses **flock**(2) to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of **lpd** for the programs **lpq**(1) and **lprm**(1).

Control Files

After the daemon has successfully set the lock, it scans the directory for files beginning with **cf**. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character that indicates what to do with the remainder of the line.

- **J** Job name to print on the burst page.
- **C** Classification line on the burst page.
- **L** Literal. This line contains identification information from the password file, and causes a burst page to be printed.
- **T** Title string for page headings printed by **pr**(1V).
- **H** Hostname of the machine where **lpr**(1) was invoked.
- **P** Person. Login name of the person who invoked **lpr**(1). This is used to verify ownership by **lprm**(1).
- **M** Send mail to the specified user when the current print job completes.
- **f** Formatted File, the name of a file to print that is already formatted.
- **l** Like **f**, but passes control characters along, and does not make page breaks.
- **p** Name of a file to print using **pr**(1V) as a filter.
- **t** Troff File. The file contains **troff**(1) output (cat phototypesetter commands).

n	Ditroff File. The file contains device independent troff output.
d	DVI File. The file contains $\TeX$ output (DVI format from Stanford).
g	Graph File. The file contains data produced by **plot**(3X).
c	Cifplot File. The file contains data produced by *cifplot*.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.
1	Troff Font R. The name of a font file to use instead of the default.
2	Troff Font I. The name of the font file to use instead of the default.
3	Troff Font B. The name of the font file to use instead of the default.
4	Troff Font S. The name of the font file to use instead of the default.
W	Width. Changes the page width (in characters) used by **pr**(1V) and the text filters.
I	Indent. Specify the number of characters by which to indent the output.
U	Unlink. The name of file to remove upon completion of printing.
N	Filename. The name of the file being printed, or a blank for the standard input (when **lpr**(1) is invoked in a pipeline).

Data Files

When a file is spooled for printing, the contents are copied into a data file in the spool directory. Data file names begin with **df**. When **lpr** is called with the **-s** option, the control files contain a symbolic link to the actual file, and no data files are created.

Minfree File

The file *minfree* in each spool directory contains the number of kilobytes to leave free so that the line printer queue won't completely fill the disk.

FILES

/etc/printcap	printer description file
/var/spool/*	spool directories
/var/spool/*/minfree	minimum free space to leave
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	hosts allowed equivalent host access
/etc/hosts.lpd	hosts allowed printer access only

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **hosts**(5), **hosts.equiv**(5), **printcap**(5), **lpc**(8), **pac**(8)

NAME

mailstats – print statistics collected by sendmail

SYNOPSIS

/usr/etc/mailstats [*filename*]

DESCRIPTION

mailstats prints out the statistics collected by the **sendmail** program on mailer usage. These statistics are collected if the file indicated by the S configuration option of **sendmail** exists. The **mailstats** program first prints the time that the statistics file was created and the last time it was modified. It will then print a table with one row for each mailer specified in the configuration file. The first column is the mailer number, followed by the symbolic name of the mailer. The next two columns refer to the number of messages received by *sendmail*, and the last two columns refer to messages sent by *sendmail*. The number of messages and their total size (in 1024 byte units) is given. No numbers are printed if no messages were sent (or received) for any mailer.

You might want to add an entry to **/var/spool/cron/crontab/root** to reinitialize the statistics file once a night. Copy **/dev/null** into the statistics file or otherwise truncate it to reset the counters.

FILES

/etc/sendmail.st	default statistics file
/etc/sendmail.cf	sendmail configuration file
/var/spool/cron/crontab/root	
/dev/null	

SEE ALSO

sendmail(8)

BUGS

Mailstats should read the configuration file instead of having a hard-wired table mapping mailer numbers to names.

NAME

makedbm – make a NIS ndbm file

SYNOPSIS

/usr/etc/yp/makedbm [**–b**] [**–l**] [–s] [**–i** *yp_input_file*] [**–o** *yp_output_name*]
[**–d** *yp_domain_name*] [**–m** *yp_master_name*] *infile outfile*

makedbm [**–u** *dbmfilename*]

DESCRIPTION

makedbm takes *infile* and converts it to a pair of files in **ndbm**(3) format, namely *outfile*.**pag** and *outfile*.**dir**. Each line of the input file is converted to a single **dbm** record. All characters up to the first TAB or SPACE form the key, and the rest of the line is the data. If a line ends with '\', then the data for that record is continued on to the next line. It is left for the clients of the Network Interface Service (NIS) to interpret #; **makedbm** does not itself treat it as a comment character. *infile* can be '–', in which case the standard input is read.

makedbm is meant to be used in generating **dbm** files for the NIS service, and it generates a special entry with the key *yp_last_modified*, which is the date of *infile* (or the current time, if *infile* is '–').

OPTIONS

–b Interdomain. Propagate a map to all servers using the interdomain name server **named**(8C).

–l Lowercase. Convert the keys of the given map to lower case, so that host name matches, for example, can work independent of upper or lower case distinctions.

–s Secure map. Accept connections from secure NIS networks only.

–i *yp_input_file*
Create a special entry with the key *yp_input_file*.

–o *yp_output_name*
Create a special entry with the key *yp_output_name*.

–d *yp_domain_name*
Create a special entry with the key *yp_domain_name*.

–m *yp_master_name*
Create a special entry with the key *yp_master_name*. If no master host name is specified, *yp_master_name* will be set to the local host name.

–u *dbmfilename*
Undo a **dbm** file. That is, print out a **dbm** file one entry per line, with a single space separating keys from values.

EXAMPLE

It is easy to write shell scripts to convert standard files such as **/etc/passwd** to the key value form used by **makedbm**. For example:

```
#!/bin/awk -f
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

takes the **/etc/passwd** file and converts it to a form that can be read by **makedbm** to make the NIS file **passwd.byname**. That is, the key is a username, and the value is the remaining line in the **/etc/passwd** file.

SEE ALSO

yppasswd(1), **ndbm**(3), **named**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

makedev, MAKEDEV – make system special files

SYNOPSIS

/dev/MAKEDEV *device-name* ...

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the **/dev** directory, as this is the normal location of special files. Arguments to **MAKEDEV** are usually of the form *device-name?* where *device-name* is one of the supported devices listed in section 4 of the manual and '*?*' is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; for example, **/dev/console**, **/dev/tty**.

local Create those devices specific to the local site. This request runs the shell file **/dev/MAKEDEV.local**. Site specific commands, such as those used to setup dialup lines as "ttyd?" should be included in this file.

Since all devices are created using **mknod**(8), this shell script is useful only to the super-user.

FILES

/dev/console /dev/MAKEDEV.local /dev/tty

SEE ALSO

intro(4), **config**(8), **mknod**(8)

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use **sh –x MAKEDEV** in case of trouble.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

makekey is intended for programs that perform encryption (for instance, **ed**(1) and **crypt**(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), **ed**(1)

NAME

mc68881version – print the MC68881 mask number and approximate clock rate

SYNOPSIS

/usr/etc/mc68881version

AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only.

DESCRIPTION

mc68881version determines whether an MC68881 or MC68882 floating-point coprocessor is available, and if so, determines its apparent mask number and approximate clock rate and prints them on the standard output. The reported clock rate is derived by timing floating-point operations with **getrusage**(2) and is thus somewhat variable; best results may be obtained in single-user mode. The same applies to the differentiation between MC68881 and MC68882 ; these can be distinguished in user mode only by timing tests.

SEE ALSO

getrusage(2)

NAME

mconnect – connect to SMTP mail server socket

SYNOPSIS

/usr/etc/mconnect [**–p** *port*] [**–r**] [*hostname*]

DESCRIPTION

mconnect opens a connection to the mail server on a given host, so that it can be tested independently of all other mail software. If no host is given, the connection is made to the local host. Servers expect to speak the Simple Mail Transfer Protocol (SMTP) on this connection. Exit by typing the **quit** command. Typing EOF will send an end of file to the server. An interrupt closes the connection immediately and exits.

OPTIONS

–p *port* Specify the port number instead of the default SMTP port (number 25) as the next argument.

–r ''Raw'' mode: disable the default line buffering and input handling. This gives you a similar effect as **telnet** to port number 25, not very useful.

FILES

/usr/lib/sendmail.hf help file for SMTP commands

SEE ALSO

sendmail(8)

Postel, Jonathan B *Simple Mail Transfer Protocol*, RFC821 August 1982, SRI Network Information Center

NAME

mkfile – create a file

SYNOPSIS

mkfile [**−nv**] *size*[**k** | **b** | **m**] *filename* ...

DESCRIPTION

mkfile creates one or more files that are suitable for use as NFS-mounted swap areas, or as local swap areas. The sticky bit is set, and the file is padded with zeroes by default. The default *size* is in bytes, but it can be flagged as kilobytes, blocks, or megabytes, with the **k**, **b**, or **m** suffixes, respectively.

OPTIONS

−n Create an empty *filename*. The size is noted, but disk blocks aren't allocated until data is written to them.

−v Verbose. Report the names and sizes of created files.

SEE ALSO

swapon(2), **fstab**(5), **swapon**(8)

NAME

mkfs – construct a file system

SYNOPSIS

/usr/etc/mkfs [**–N**] *special size* [*nsect*] [*ntrack*] [*blksize*] [*fragsize*] [*ncpg*] [*minfree*]
[*rps*] [*nbpi*] [*opt*] [*apc*] [*rot*] [*nrpos*]

DESCRIPTION

Note: file systems are normally created with the **newfs**(8) command.

mkfs constructs a file system by writing on the special file *special* unless the –**N** flag has been specified. *special* must be specified as a raw device and disk partition. For example, to create a file system on **sd0**, specify **/dev/rsd0[a-h]**, where **a-h** is the disk partition.

The numeric *size* specifies the number of sectors in the file system. **mkfs** builds a file system with a root directory and a lost+found directory (see **fsck**(8)). The number of inodes is calculated as a function of the file system size. No boot program is initialized by **mkfs** (see **newfs**(8)).

You must be super-user to use this command.

OPTIONS

–N Print out the file system parameters without actually creating the file system.

The following arguments allow fine tune control over the parameters of the file system.

nsect The number of sectors per track on the disk. The default is **32**.

ntrack The number of tracks per cylinder on the disk. The default is **16**.

blksize The primary block size for files on the file system. It must be a power of two, currently selected from **4096** or **8192** (the default).

fragsize The fragment size for files on the file system. The *fragsize* represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range **512** to **8192**. The default is **1024**.

ncpg The number of disk cylinders per cylinder group. The default is **16**.

minfree The minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is **10%**.

rps The rotational speed of the disk, in revolutions per second. The default is **60**.

nbpi The number of bytes for which one inode block is allocated. This parameter is currently set at one inode block for every 2048 bytes.

opt Space or time optimization preference; **s** specifies optimization for space, **t** specifies optimization for time. The default is **t**.

apc The number of alternates per cylinder (SCSI devices only). The default is **0**.

rot The expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

nrpos The number of distinguished rotational positions. The default is 8.

Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

dir(5), **fs**(5), **fsck**(8), **newfs**(8), **tunefs**(8)

System and Network Administration
McKusick, Joy, Leffler; *A Fast File System for UNIX*

NOTES

newfs(8) is preferred for most routine uses.

NAME

mknod – build special file

SYNOPSIS

/usr/etc/mknod *filename* [**c**] [**b**] *major minor*

/usr/etc/mknod *filename* **p**

DESCRIPTION

mknod makes a special file. The first argument is the *filename* of the entry. In the first form, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number). Only the super-user is permitted to invoke this form of the **mknod** command.

In the second form, **mknod** makes a named pipe (FIFO).

The first form of **mknod** is only for use by system configuration people. Normally you should use **/dev/MAKEDEV** instead when making special files.

SEE ALSO

mknod(2V), **makedev**(8)

NAME

mkproto – construct a prototype file system

SYNOPSIS

/usr/etc/mkproto *special proto*

DESCRIPTION

mkproto is used to bootstrap a new file system. First a new file system is created using **newfs**(8). **mkproto** is then used to copy files from the old file system into the new file system according to the directions found in the prototype file **proto**. The prototype file contains tokens separated by SPACE or NEWLINE characters. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **–bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or '–' to specify set-user-id mode or not. The third is **g** or '–' for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see **chmod**(1V).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, **mkproto** makes the entries '**.**' and '**..**' and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /usr/bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO

chmod(1V), **fs**(5), **dir**(5), **fsck**(8), **newfs**(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

modload – load a module

SYNOPSIS

modload *filename* [**–conf** *config_file*] [**–entry** *entry_point*] [**–exec** *exec_file*] [**–o** *output_file*] [**–nolink**] [**–A** *vmunix_file*]

DESCRIPTION

modload loads a loadable module into a running system. The input file *filename* is an object file (**.o** file).

OPTIONS

–conf *config_file*
: Use this configuration file to configure the loadable driver being loaded. The commands in this file are the same as those that the **config**(8) program recognizes. There are two additional commands, **blockmajor** and **charmajor**, shown in the configuration file example below.

–entry *entry_point*
: This is the module entry point. This is passed by **modload** to **ld**(1) when the module is linked. The default module entry point name is '*xxx* **init**'.

–exec *exec_file*
: This is the name of a shell script or executable image file that is executed if the module is successfully loaded. It is always passed the module id and module type as the first two arguments. For loadable drivers, the third and fourth arguments are the block major and character major numbers respectively. For a loadable system call, the third argument is the system call number.

–o *output_file*
: This is the name of the output file that is produced by the linker. If this option is omitted, then the output file name is *filename*> without the '**.o**'.

–nolink This option can be used if **modload** has already been issued once and the output file already exists. One must take care that neither the kernel nor the module have changed.

–A *vmunix_file*
: This is the file that is passed to the linker to resolve module references to kernel symbols. The default is **/vmunix**. The symbol file must be for the currently running kernel or the module is likely to crash the system.

EXAMPLES

```
controller      fdc0 at atmem csr 0x001000 irq 6 priority 3
controller      fdc2 at atmem csr 0x002000 irq 5 priority 2
disk            fd0 at fdc0 drive 0
disk            fd0 at fdc0 drive 1
disk            fd0 at fdc0 drive 2
device          fd0 at fdc2 drive 0 csr 0x003000 irq 4 priority 2
disk            fd0 at fdc2 drive 1
blockmajor 51
charmajor 52
```

SEE ALSO

ld(1), **modunload**(8), **modstat**(8)

NAME

modstat – display status of loadable modules

SYNOPSIS

modstat [**–id** *module_id*]

DESCRIPTION

modstat displays the status of the loaded modules.

OPTIONS

–id *module_id*

Display the status of only this module.

SEE ALSO

modload(8), **modunload**(8)

NAME

modunload – unload a module

SYNOPSIS

modunload –id *module_id* [**–exec exec_file**]

DESCRIPTION

modunload unloads a loadable module from a running system. The *module_id* is the ID of the module as shown by **modstat**(8).

OPTIONS

–exec *exec_file*

This is the name of a shell script or executable image file that will be executed before the module is unloaded. It is always passed the module ID and module type as the first two arguments. For loadable drivers, the third and fourth arguments are the block major and character major numbers respectively. For a loadable system call, the third argument is the system call number.

SEE ALSO

modload(8), **modstat**(8)

NAME
monitor – system ROM monitor

SYNOPSIS
L1–A

BREAK

DESCRIPTION
The CPU board of the Sun workstation contains an EPROM (or set of EPROMs), called the *monitor*, that controls the system during startup. The monitor tests the system before attempting to boot the operating system. If you interrupt the boot procedure by holding down **L1** while typing **a** or **A** on the workstation keyboard (or **BREAK** if the console is a dumb terminal) the monitor issues the prompt:

>

and accepts commands interactively.

USAGE
Modes
The monitor supports three security modes (non-secure, command secure, and fully secure) and an authentication password. Access to monitor commands is controlled by these security modes. In **non-secure** mode all monitor commands are allowed. In **command secure** mode, only the **b**(boot) command with no arguments and the **c**(continue) command with no arguments may be entered without supplying the authentication password. In **fully secure** mode, only the **c**(continue) command with no arguments may be entered without supplying the authentication password. Note: The system will not auto-reboot in fully secure mode. The authentication password must be entered before booting will take place.

Commands
+|– Increment or decrement the current address and display the contents of the new location.

^C *source destination n*
(caret-C) Copy, byte-by-byte a block of length *n* from the *source* address to the *destination* address.

^I *program* (caret-I) Display the compilation date and location of *program*.

^T *virtual_address*
(caret-T) Display the physical address to which *virtual_address* is mapped.

a [*n*] [*action*]... (Sun-2 and Sun-3 systems only)
Open A–register (cpu address register) *n*, and perform indicated actions. The number *n* can be any value from **0** to **7**, inclusive. The default value is **0**. A hexadecimal *action* argument assigns the value you supply to the register *n*. A non-hex *action* terminates command input.

b [**!**] [*device* [**(** *c,u,p* **)**]] [*pathname*] [*arguments_list*]
b[?] Reset appropriate parts of the system and bootstrap a program. A '**!**' (preceding the *device* argument) prevents the system reset from occurring. Programs can be loaded from various devices (such as a disk, tape or Ethernet). '**b**' with no arguments will cause a default boot, either from a disk, or from an Ethernet controller. '**b?**' displays all boot devices and their *device* arguments, where *device* is one of:

- **ie** Intel Ethernet
- **le** Lance Ethernet (Sun-2, Sun-3, Sun-4 systems only)
- **sd** SCSI disk
- **st** SCSI 1/4" tape
- **mt** Tape Master 9-track 1/2" tape (Sun-2, Sun-3, Sun-4 systems only)
- **xd** Xylogics 7053 disk (Sun-2, Sun-3, Sun-4 systems only)
- **xt** Xylogics 1/2" tape (Sun-2, Sun-3, Sun-4 systems only)
- **xy** Xylogics 440/450 disk (Sun-2, Sun-3, Sun-4 systems only)
- **fd** Diskette (Sun386i system only)

c A controller number (**0** if only one controller),

u A unit number (**0** if only one driver), and

p A partition.

pathname A pathname for a program such as **/stand/diag. /vmunix** is the default.

arguments_list
A list of up to seven arguments to pass to the program being booted.

c [*virtual_address*]
Resume execution of a program. When given, *virtual_address* is the address at which execution will resume. The default is the current PC (EIP on Sun386i systems). Registers are restored to the values shown by the **a**, **d**, and **r** commands (for Sun-2 and Sun-3 systems), or by the **d** and **r** commands (for Sun-4 systems), or by the **d** command (for Sun386i systems).

d [*window_number*] (Sun-4 systems only)
Display (dump) the state of the processor. The processor state is observable only after:

- An unexpected trap was encountered.
- A user program dropped into the monitor (by calling *abortent*).
- The user manually entered the monitor by typing **L1–A** or **BREAK**.

The display consists of the following:

- The special registers: PSR, PC, nPC, TBR, WIM and Y
- Eight global registers, and
- 24 window registers (8 *in*, 8 *local*, and 8 *out*), corresponding to one of the 7 available windows. If a Floating-Point Unit is on board, its status register along with its 32 floating-point registers are also shown.

window_number
Display the indicated *window_number*, which can be any value between 0 and 6, inclusive. If no window is specified and the PSR's current window pointer contains a valid window number, registers from the window that was active just prior to entry into the monitor are displayed. Otherwise, registers from window 0 are displayed.

d (Sun386i systems only)
Display (dump) the state of the processor. This display consists of the registers, listed below:

Processor Registers:	EAX, ECX, EDX, ESI, EDI, ESP, EBP, EFLAGS, EIP
Segment Registers:	ES, CS, SS, DS, FS, GS
Memory Management Registers:	GDTR, LDTR, IDTR, TR
Control Registers:	CR0, CR2, CR3
Debug Registers:	DR0, DR1 , DR2 , DR3, DR6, DR7
Test Registers:	TR6, TR7

The processor's state is observable only after an unexpected trap, a user program has "dropped" into the monitor (by calling monitor function abortentor) or the user has manually "broken " into the monitor (by typing **L1–A** on the Workstation console, or **BREAK** on the dumb terminal's keyboard.

d [*n*] [*action*]... (Sun-2 and Sun-3 systems only)
Open **D**–register (cpu data register) *n*, and perform indicated actions. The number *n* can be any value from **0** to **7**, inclusive. The default is **0**. See the **a** command for a description of *action*.

e [*virtual_address*] [*action*] . . .
Open the 16 bit word at *virtual_address* (default zero). On Sun-2, Sun-3, and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. See the **a** command for a description of *action*.

f *virtual_address1 virtual_address2 pattern* [*size*] (Sun-3 and Sun-4 systems only)
Fill the bytes, words or long words from *virtual_address1* (lower) to *virtual_address2* (higher) with the constant, *pattern*. The *size* argument can take one of the following values

- **b** byte format (the default)
- **w** word format
- **l** long word format

For example, the following command fills the address block from 0x1000 to 0x2000 with the word pattern, 0xABCD:

f 1000 2000 ABCD W

g [*vector*] [*argument*]
g [*virtual_address*] [*argument*]
Goto (jump to) a predetermined or default routine (first form), or to a user-specified routine (second form). The value of *argument* is passed to the routine. If the *vector* or *virtual_address* argument is omitted, the value in the PC is used as the address to jump to.

To set up a predetermined routine to jump to, a user program must, prior to executing the monitor's **g** command, set the variable ***romp->v_vector_cmd** to be equal to the virtual address of the desired routine. Predetermined routines need not necessarily return control to the monitor.

The default routine, defined by the monitor, prints the user-supplied *vector* according to the format supplied in *argument*. This format can be one of:

%x hexadecimal
%d decimal

g0 (Sun-2, Sun-3, and Sun-4 only)
When the monitor is running as a result of the system being interrupted, force a panic and produce a crash dump.

g4 When the monitor is running as a result of the system being interrupted, force a kernel stack trace.

h (Sun-3 and Sun-4 and Sun386i systems)
Display the help menu for monitor commands and their descriptions. To return to the monitor's basic command level, press ESCAPE or **q** before pressing RETURN.

i [*cache_data_offset*] [*action*] . . . (Sun-3/200 series and Sun-4 systems only)
Modify cache data RAM command. Display and/or modify one or more of the
Modify cache data RAM command. Display and/or modify one or more of the cache data addresses. See the **a** command for a description of *action*.

j [*cache_tag_offset*] [*action*] . . . (Sun-3/200 series and Sun-4 systems only)
Modify cache tag RAM command. Display and/or modify the contents of one or more of the cache tag addresses. See the **a** command for a description of *action*.

k [*reset_level*]
Reset the system. If *reset_level* is:

- **0** CPU reset only (Sun-2 and Sun-3 systems). Reset VMEbus, interrupt registers, video monitor (Sun-4 systems). This is the default. Reset video (Sun386i systems).
- **1** Software reset.

2 Power-on reset. Resets and clears the memory. Runs the EPROM-based diagnostic self test, which can take several minutes, depending upon how much memory is being tested.

kb Display the system banner.

l [*virtual_address*] [*action*] . . .
Open the long word (32 bit) at memory address *virtual_address* (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command (below). See the **a** command for a description of *action*.

m [*virtual_address*] [*action*] . . .
Open the segment map entry that maps *virtual_address* (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. Not supported on Sun386i. See the **a** command for a description of *action*.

nd (Sun386i systems only)
ne
ni Disable, enable, or invalidate the cache, respectively

o [*virtual_address*] [action*]* . . .
Open the byte location specified by
virtual_address (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. See the **a** command for a description of *action*.

p [*virtual_address*] [*action*]. . .
Open the page map entry that maps *virtual_address* (default zero) in the address space defined by the **s** command. See the **a** command for a description of *action*.

p [*port_address*] [[*nonhex_char* [*hex_value*] | *hex_value*] ...] (Sun386i systems only)
Display or modify the contents of one or more port I/O addresses in byte mode. Each port address is treated as a 8-bit unit. The optional *port_address*, argument, which is a 16-bit quantity, specifies the initial port I/O address. See the **e** command for argument descriptions.

q [*eeprom_offset*] [*action*]. . . (Sun-3 and Sun-4 systems only)
Open the EEPROM *eeprom_offset* (default zero) in the EEPROM address space. All addresses are referenced from the beginning or base of the EEPROM in physical address space, and a limit check is performed to insure that no address beyond the EEPROM physical space is accessed. On Sun386i systems, open the NVRAM *nvram_offset* (default zero). This command is used to display or modify configuration parameters, such as: the amount of memory to test during self test, whether to display a standard or custom banner, if a serial port (A or B) is to be the system console, etc. See the **a** command for a description of *action*.

r [*reg_name*] [[*nonhex_char* [*hex_value*] | *hex_value*] ...] (Sun386i systems only)
Display or modify one or more of the processor registers. If *reg_name* is specified (2 or 3 characters from the above list), that register is displayed first. The default is EAX. See note on register availability under the command **d** (for Sun386i systems). See the **e** command for argument descriptions.

s [*step_count*] (Sun386i systems only)
Single step the execution of the interrupted program. The *step_count* argument specifies the number of single steps to execute before displaying the monitor prompt. The default is 1.

r [*register_number*] [*action*] . . . (Sun-2 and Sun-3 systems only)
Display and/or modify the register indicated. *register_number* can be one of:

CA 68020 Cache Address Register
CC 68020 Cache Control Register
CX 68020 System and User Context

DF	Destination Function code
IS	68020 Interrupt Stack Pointer
MS	68020 Master Stack Pointer
PC	Program Counter
SC	68010 System Context
SF	Source Function code
SR	Status Register
SS	68010 Supervisor Stack Pointer
UC	68010 User Context
US	User Stack Pointer
VB	Vector Base

Alterations to these registers (except SC and UC) do not take effect until the next **c** command is executed. See the **a** command for a description of *action*.

r [*register_number*] (Sun-4 systems only)
r [*register_type*]
r [*w window_number*]

Display and/or modify one or more of the IU or FPU registers.

A hexadecimal *register_number* can be one of:

0x00—0x0f	window(0,i0)—window(0,i7), window(0,i0)—window(0,i7)
0x16—0x1f	window(1,i0)—window(1,i7), window(1,i0)—window(1,i7)
0x20—0x2f	window(2,i0)—window(2,i7), window(2,i0)—window(2,i7)
0x30—0x3f	window(3,i0)—window(3,i7), window(3,i0)—window(3,i7)
0x40—0x4f	window(4,i0)—window(4,i7), window(4,i0)—window(4,i7)
0x50—0x5f	window(5,i0)—window(5,i7), window(5,i0)—window(5,i7)
0x60—0x6f	window(6,i0)—window(6,i7), window(6,i0)—window(6,i7)
0x70—0x77	g0, g1, g2, g3, g4, g5, g6, g7
0x78—0x7d	PSR, PC, nPC, WIM, TBR, Y
0x7e—0x9e	FSR, f0—f31

Register numbers can only be displayed after an unexpected trap, a user program has entered the monitor using the *abortent* function, or the user has entered the monitor by manually typing **L1–A** or **BREAK**.

If a *register_type* is given, the first register of the indicated type is displayed. *register_type* can be one of:

f	floating-point
g	global
s	special

If **w** and a *window_number* (**0—6**) are given, the first *in*-register within the indicated window is displayed. If *window_number* is omitted, the window that was active just prior to entering the monitor is used. If the PSR's current window pointer is invalid, window 0 is used.

s [*code*] (Sun-2 and Sun-3 systems only)
Set or query the address space to be used by subsequent memory access commands. *code* is one of:

0	undefined
1	user data space
2	user program space
3	user control space
4	undefined
5	supervisor data space
6	supervisor program space
7	supervisor control space

If *code* is omitted, **s** displays the current address space.

s [*asi*] (Sun-4 systems only)
Set or display the Address Space Identifier. With no argument, **s** displays the current Address Space Identifier. The *asi* value can be one of:

0x2	control space
0x3	segment table
0x4	Page table
0x8	user instruction
0x9	supervisor instruction
0xa	user data
0xb	supervisor data
0xc	flush segment
0xd	flush page
0xe	flush context
0xf	cache data

t [*program*] (Sun-3 systems only)
Trace the indicated standalone *program*. Works only with programs that do not affect interrupt vectors.

u [*echo*]
u [*port*] [*options*] [*baud_rate*]
u [**u**] [*virtual_address*]
With no arguments, display the current I/O device characteristics including: current input device, current output device, baud rates for serial ports A and B, an input-to-output echo indicator, and virtual addresses of mapped UART devices. With arguments, set or configure the current I/O device. With the **u** argument (**uu**...), set the I/O device to be the *virtual_address* of a UART device currently mapped.

echo Can be either **e** to enable input to be echoed to the output device, or **ne**, to indicate that input is not echoed.

port Assign the indicated *port* to be the current I/O device. *port* can be one of:

a	serial port A
b	serial port B (except on Sun386i systems)
k	the workstation keyboard
s	the workstation screen

baud_rate Any legal baud rate.

options can be any combination of:

i	input
o	output

u UART
e echo input to output
ne do not echo input
r reset indicated serial port (**a** and **b** ports only)

If either **a** or **b** is supplied, and no *options* are given, the serial port is assigned for both input and output. If **k** is supplied with no options, it is assigned for input only. If s is supplied with no options, it is assigned for output only.

v *virtual_address1 virtual_address2* [*size*] (Sun-3 and Sun-4 systems only)

Display the contents of *virtual_address1* (lower) *virtual_address2* (higher) in the format specified by *size*:

b byte format (the default)
w word format
l long word format

Enter return to pause for viewing; enter another return character to resume the display. To terminate the display at any time, press the space bar.

For example, the following command displays the contents of virtual address space from address 0x1000 to 0x2000 in word format:

v 1000 2000 W

w [*virtual_address*] [*argument*] (Sun-3 and Sun-4 systems only)

Set the execution vector to a predetermined or default routine. Pass *virtual_address* and *argument* to that routine.

To set up a predetermined routine to jump to, a user program must, prior to executing the monitor's w command, set the variable ***romp->v_vector_cmd** to be equal to the virtual address of the desired routine. Predetermined routines need not necessarily return control to the monitor.

The default routine, defined by the monitor, prints the user-supplied *vector* according to the format supplied in *argument*. This format can be one of:

%x hexadecimal
%d decimal

x (Sun-3 and Sun-4 systems only)

Display a menu of extended tests. These diagnostics permit additional testing of such things as the I/O port connectors, video memory, workstation memory and keyboard, and boot device paths.

y c *context_number* (Sun-4 systems only)
y p|s *context_number virtual_address*

Flush the indicated context, context page, or context segment.

c flush context *context_number*
p flush the page beginning at *virtual_address* within context *context_number*
s flush the segment beginning at *virtual_address* within context *context_number*

z [*number*] [*breakpoint_virtual_address* [*type*] [*len*]] (Sun386i systems only)

Set or reset breakpoints for debugging. With no arguments, this command displays the existing breakpoints. The *number* argument is a values from 0 to 3, corresponding to the processor debug registers, **DR0** to **DR3**, respectively. Up to 4 distinct breakpoints can be specified. If *number* is not specified then the monitor chooses a breakpoint number. The *breakpoint_virtual_address* argument specifies the breakpoint address. The *type* argument can be one of:

x Instruction Execution breakpoint (the default)
m for Data Write only breakpoint
r Data Reads and Writes only breakpoint.

The *len* argument can be one of: '**b**', '**w**', or '**l**', corresponding to the breakpoint field length of byte, word, or long-word, respectively. The default is '**b**'. Since the breakpoints are set in the on-chip registers, an instruction breakpoint can be placed in ROM code or in code shared by several tasks. If the *number* argument is specified but not *breakpoint_virtual_address*, the corresponding breakpoint is reset.

z [*virtual_address*] (Sun-3 systems only)

Set a breakpoint at *virtual_address* in the address space selected by the s command.

FILES

/vmunix

SEE ALSO

eeprom(8S)

NAME

mount, umount – mount and unmount file systems

SYNOPSIS

/usr/etc/mount [**–p**]
/usr/etc/mount –a [**fnv**] [**–t** *type*]
/usr/etc/mount [**–fnrv**] [**–t** *type*] [**–o** *options*] *filesystem directory*
/usr/etc/mount [**–vfn**] [**–o** *options*] *filesystem* | *directory*
/usr/etc/mount –d [**fnvr**] [**–o** *options*] *RFS-resource* | *directory*

/usr/etc/umount [**–t** *type*] [**–h** *host*]
/usr/etc/umount –a [**v**]
/usr/etc/umount [**–v**] *filesystem* | *directory* ...
/usr/etc/umount [**–d**] *RFS-resource* | *directory*

DESCRIPTION

mount attaches a named *filesystem* to the file system hierarchy at the pathname location *directory*, which must already exist. If *directory* has any contents prior to the **mount** operation, these remain hidden until the *filesystem* is once again unmounted. If *filesystem* is of the form *host:pathname*, it is assumed to be an NFS file system (type **nfs**).

umount unmounts a currently mounted file system, which can be specified either as a *directory* or a *filesystem*.

mount and **umount** maintain a table of mounted file systems in **/etc/mtab**, described in **fstab**(5). If invoked without an argument, **mount** displays the contents of this table. If invoked with either a *filesystem* or *directory* only, **mount** searches the file **/etc/fstab** for a matching entry, and mounts the file system indicated in that entry on the indicated directory.

mount also allows the creation of new, virtual file systems using **loopback mounts**. Loopback file systems provide access to existing files using alternate pathnames. Once a virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original file system, however, **are** visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another file system hierarchy. Rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

The standard RC files first perform **4.2** mounts, then **nfs** mounts, during booting. On Sun386i systems, **lo** (loopback) mounts are performed just after **4.2** mounts. **/etc/fstab** files depending on alternate mount orders at boot time will fail to work as expected. Manual modification of **/etc/rc.local** will be needed to make such mount orders work.

See **lofs**(4S) and **fstab**(5) for more information and WARNINGS about loopback mounts.

OPTIONS

mount

–p Print the list of mounted file systems in a format suitable for use in **/etc/fstab**.

–a All. Attempt to mount all the file systems described in **/etc/fstab**. If a *type* argument is specified with **–t**, mount all file systems of that type. Using **–a**, **mount** builds a dependency tree of mount points in **/etc/fstab**. **mount** will correctly mount these file systems regardless of their order in **/etc/fstab** (except loopback mounts; see WARNINGS below).

–f Fake an **/etc/mtab** entry, but do not actually mount any file systems.

–n Mount the file system without making an entry in **/etc/mtab**.

–v Verbose. Display a message indicating each file system being mounted.

−t *type* Specify a file system type. The accepted types are **4.2**, **nfs**, **rfs**, **lo**, **hsfs**, and **tmp**. See **fstab**(5) for a description of **4.2**, **hsfs**, and **nfs**; see **lofs**(4S) for a description of **lo**; and see **tmpfs**(4) for a description of **tmp**. See *System and Network Administration* for details on **rfs**.

−r Mount the specified file system read-only, even if the entry in **/etc/fstab** specifies that it is to be mounted read-write.

Physically write-protected and magnetic-tape file systems must be mounted read-only. Otherwise errors occur when the system attempts to update access times, even if no write operation is attempted.

−d Mount an RFS file system. This option provides compatibility with the System V, Release 3 syntax for RFS mounts. Alternatively, the equivalent Sun syntax, **−t rfs**, may be used.

−o *options*

Specify file system *options*, a comma-separated list of words from the list below. Some options are valid for all file system types, while others apply to a specific type only.

options valid on *all* file systems:

rw|**ro**	Read/write or read-only.
suid|**nosuid**	Setuid execution allowed or disallowed.
grpid	Create files with BSD semantics for the propagation of the group ID. Under this option, files inherit the GID of the directory in which they are created, regardless of the directory's set-GID bit.
noauto	Do not mount this file system that is currently mounted read-only. If the file system is not currently mounted, an error results.
remount	If the file system is currently mounted, and if the entry in **/etc/fstab** specifies that it is to be mounted read-write or **rw** was specified along with **remount**, remount the file system making it read-write. If the entry in **/etc/fstab** specifies that it is to be mounted read-only and **rw** was not specified, the file system is not remounted. If the file system is currently mounted read-write, specifying **ro** along with **remount** results in an error. If the file system is not currently mounted, an error results.

The default is '**rw, suid**'.

options specific to **4.2** file systems:

quota|**noquota**	Usage limits are enforced, or are not enforced. The default is **noquota**.

options specific to **nfs** (NFS) file systems:

bg|**fg**	If the first attempt fails, retry in the background, or, in the foreground.
noquota	Prevent **quota**(1) from checking whether the user is over quota on this file system; if the file system has quotas enabled on the server, quotas will still be checked for operations on this file system.
retry=*n*	The number of times to retry the mount operation.
rsize=*n*	Set the read buffer size to *n* bytes.
wsize=*n*	Set the write buffer size to *n* bytes.
timeo=*n*	Set the NFS timeout to *n* tenths of a second.
retrans=*n*	The number of NFS retransmissions.
port=*n*	The server IP port number.
soft|**hard**	Return an error if the server does not respond, or continue the retry request until the server responds.
intr	Allow keyboard interrupts on hard mounts.
secure	Use a more secure protocol for NFS transactions.
posix	Request POSIX.1 semantics for the file system. Requires a mount version 2 **mountd**(8C) on the server.

acregmin=*n*	Hold cached attributes for at least *n* seconds after file modification.
acregmax=*n*	Hold cached attributes for no more than *n* seconds after file modification.
acdirmin=*n*	Hold cached attributes for at least *n* seconds after directory update.
acdirmax=*n*	Hold cached attributes for no more than *n* seconds after directory update.
actimeo=*n*	Set *min* and *max* times for regular files and directories to *n* seconds.
nocto	Suppress fresh attributes when opening a file.
noac	Suppress attribute and name (lookup) caching.

Regular defaults are:

fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,
acregmin=3,acregmax=60,acdirmin=30,acdirmax=60

actimeo has no default; it sets **acregmin**, **acregmax**, **acdirmin** and **acdirmax**

Defaults for **rsize** and **wsize** are set internally by the system kernel.

options specific to **rfs** (**RFS**) file systems:

bg | fg	If the first attempt fails, retry in the background, or, in the foreground.
retry=*n*	The number of times to retry the mount operation.

Defaults are the same as for NFS.

umount

−h *host* Unmount all file systems listed in **/etc/mtab** that are remote-mounted from *host*.

−t *type* Unmount all file systems listed in **/etc/mtab** that are of a given *type*.

−a Unmount all file systems currently mounted (as listed in **/etc/mtab**).

−v Verbose. Display a message indicating each file system being unmounted.

−d Unmount an RFS file system. This option provides compatibility with the System V, Release 3 syntax for unmounting an RFS file system.

NFS FILESYSTEMS

Background vs. Foreground

Filesystems mounted with the **bg** option indicate that **mount** is to retry in the background if the server's mount daemon (mountd(8C)) does not respond. **mount** retries the request up to the count specified in the **retry=***n* option. Once the file system is mounted, each NFS request made in the kernel waits **timeo=***n* tenths of a second for a response. If no response arrives, the time-out is multiplied by **2** and the request is retransmitted. When the number of retransmissions has reached the number specified in the **retrans=***n* option, a file system mounted with the **soft** option returns an error on the request; one mounted with the **hard** option prints a warning message and continues to retry the request.

Read-Write vs. Read-Only

File systems that are mounted **rw** (read-write) should use the **hard** option.

Interrupting Processes With Pending NFS Requests

The **intr** option allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted file system.

Quotas

Quota checking on NFS file systems is performed by the server, not the client; if the file system has the **quota** option on the server, quota checking is performed for both local requests and NFS requests. When a user logs in, **login**(1) runs the **quota**(1) program to check whether the user is over their quota on any of the file systems mounted on the machine. This check is performed for NFS file systems by an RPC call to the **rquotad**(8C) server on the machine from which the file system is mounted. This can be time-consuming, especially if the remote machine is down. If the **noquota** option is specified for an NFS file system, **quota** will not check whether the user is over their quota on that file system, which can speed up the process of logging in. This does *not* disable quota checking for operations on that file system; it merely disables reporting whether the user is over quota on that file system.

Secure Filesystems

The **secure** option must be given if the server requires secure mounting for the file system.

File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting **actimeo=***n* extends flush time by *n* seconds for both regular files and directories.

SYSTEM V COMPATIBILITY

System V File-Creation Semantics

Ordinarily, when a file is created its GID is set to the effective GID of the calling process. This behavior may be overridden on a per-directory basis, by setting the set-GID bit of the parent directory; in this case, the GID is set to the GID of the parent directory (see **open**(2V) and **mkdir**(2V)). Files created on file systems that are mounted with the **grpid** option will obey BSD semantics; that is, the GID is unconditionally inherited from that of the parent directory.

EXAMPLES

To mount a local disk:
    mount /dev/xy0g /usr
To fake an entry for **nd** root:
    mount –ft 4.2 /dev/nd0 /
To mount all 4.2 file systems:
    mount –at 4.2
To mount a remote file system:
    mount –t nfs serv:/usr/src /usr/src
To mount a remote file system:
    mount serv:/usr/src /usr/src
To hard mount a remote file system:
    mount –o hard serv:/usr/src /usr/src
To mount an RFS remote file system, retrying in the background on failure:
    mount –d –o bg SRC /usr/src
To mount an RFS remote file system read-only:
    mount –t rfs –r SRC /usr/src
To save current mount state:
    mount –p > /etc/fstab
    Note: this is not recommended when running the automounter, see **automount**(8).
To loopback mount file systems:
    mount –t lo /export/tmp/localhost /tmp
    mount –t lo /export/var/localhost /var lo
    mount –t lo /export/cluster/sun386.sunos4.0.1 /usr/cluster
    mount –t lo /export/local/sun386 /usr/local

FILES

/etc/mtab	table of mounted file systems
/etc/fstab	table of file systems mounted at boot

WARNINGS

mount does not understand the mount order dependencies involved in loopback mounting. Loopback mounts may be dependent on two mounts having been previously performed, while **nfs** and **4.2** mounts are dependent only on a single previous mount. As a rule of thumb, place loopback mounts at the end of the **/etc/fstab** file. See **lofs**(4S) for a complete description.

SEE ALSO

mkdir(2V), **mount**(2V), **open**(2V), **unmount**(2V), **lofs**(4S), **fstab**(5), **mtab**(5), **automount**(8), **mountd**(8C), **nfsd**(8)

BUGS

Mounting file systems full of garbage crashes the system.

If the directory on which a file system is to be mounted is a symbolic link, the file system is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.

NAME

mountd, rpc.mountd – NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd [**–n**]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

mountd is an RPC server that answers file system mount requests. It reads the file **/etc/xtab,** described in **exports**(5), to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the **showmount**(8) command.

The **mountd** daemon is normally invoked by **rc**(8).

OPTIONS

–n Do not check that the clients are root users. Though this option makes things slightly less secure, it does allow older versions (pre-3.0) of client NFS to work.

FILES

/etc/xtab

SEE ALSO

exports(5), **rc**(8), **showmount**(8)

NAME

mount_tfs, umount_tfs – mount and dismount TFS filesystems

SYNOPSIS

/usr/etc/mount_tfs [**–r**] *fs1 fs2 … fsN dir*
/usr/etc/mount –t tfs [**–o** *options*] *fs dir*

/usr/etc/umount_tfs *dir*
/usr/etc/umount *dir*

DESCRIPTION

mount_tfs attaches a translucent file service (TFS) filesystem to the directory *dir*. After the mount, the directory *dir* is a TFS directory whose frontmost directory is *fs1* and whose backmost directory is *dir*, with any number of directories intervening. Effectively, the directories *fs1 … fsN* are stacked in front of *dir*.)

TFS filesystems can also be mounted using the **mount**(8) command. The **mount** command can only mount one directory, *fs*, in front of the backmost directory, *dir*.

umount_tfs detaches the TFS filesystem rooted at *dir*. See **tfs**(4S) for a description of a TFS filesystem.

OPTIONS

–r Mount the TFS filesystem read-only.

SEE ALSO

lsw(1), **unwhiteout**(1), **tfs**(4S), **mount**(8), **tfsd**(8)

BUGS

mount_tfs will cause **tfsd**(8) to deadlock (hang and answer no more requests) if it is used in conjunction with Network Software Environment (NSE) execsets. For example, a deadlock will occur if a user has used **mount_tfs** to mount over **/usr/lib**, and then tries to activate an NSE environment whose execset mounts over **/usr/lib**.

The directories *fs1* , *fs2* , … , *fsN* must be writable.

NAME

named, in.named – Internet domain name server

SYNOPSIS

/usr/etc/in.named [**–d** *level*] [**–p** *port*] [[**–b**] *bootfile*]

DESCRIPTION

named is the Internet domain name server. It is used by resolver libraries to provide access to the Internet distributed naming database. The domain name server is described in the *System and Network Administration.* See RFC 1034 and RFC 1035 for more details. With no arguments **named** reads **/etc/named.boot** for any initial data, and listens for queries on a privileged port.

OPTIONS

–d *level* Print debugging information. *level* is a number indicating the level of messages printed.

–p *port* Use *port* as the port number, rather than the standard port number.

–b *bootfile*
Use *bootfile* rather than **/etc/named.boot.**

EXAMPLE

```
;
;        boot file for name server
;
; type            domain            source file or host
;
primary           berkeley.edu    named.db
secondary         cc.berkeley.edu 10.2.0.78 128.32.0.10
cache             .               named.ca
```

The **primary** line states that the file **named.db** contains authoritative data for **berkeley.edu**. The file **named.db** contains data in the master file format, described in RFC 1035, except that all domain names are relative to the origin; in this case, **berkeley.edu** (see below for a more detailed description).

The **secondary** line specifies that all authoritative data under **cc.berkeley.edu** is to be transferred from the name server at **10.2.0.78**. If the transfer fails it will try **128.32.0.10**, and continue for up to 10 tries at that address. The secondary copy is also authoritative for the domain.

The **cache** line specifies that data in **named.ca** is to be placed in the cache (only used to find the root domain servers). The file **named.ca** is in the same format as **named.db**.

The master file consists of entries of the form:

$INCLUDE *<filename>*
$ORIGIN *<domain>*
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>

where *domain* is '.' for the root, '@' for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with '.', the current origin is appended to the domain. Domain names ending with '.' are unmodified.

The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero.

The *opt_class* field is currently one token, '**IN**' for the Internet.

The *type* field is one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A A host address (dotted quad).

NS An authoritative name server (domain).

MX A mail exchanger (domain).

CNAME
The canonical name for an alias (domain).

SOA Marks the start of a zone of authority (5 numbers). (see RFC 1035)).

MB A mailbox domain name (domain).

MG A mail group member (domain).

MR A mail rename domain name (domain).

NULL A null resource record (no format or data).

WKS A well know service description (not implemented yet).

PTR A domain name pointer (domain).

HINFO Host information (cpu_type OS_type).

MINFO Mailbox or mail list information (request_domain error_domain).

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process ID
/var/tmp/named.run	debug output
/var/tmp/named_dump.db	dump of the name servers database

SEE ALSO

kill(1), **signal**(3V), **resolver**(3), **resolv.conf**(5), **nslookup**(8C)

System and Network Administration

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain System Changes and Observations*, RFC 973, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

NOTES

The following signals have the specified effect when sent to the server process using the **kill**(1) command.

SIGHUP Causes server to read named.boot and reload database.

SIGINT Dumps current data base and cache to **/var/tmp/named_dump.db**.

SIGUSR1
Turns on debugging; each subsequent **SIGUSR1** increments debug level.

SIGUSR2
Turns off debugging completely.

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/usr/etc/ncheck [**–i** *numbers*] [**–as**] *filesystem*

DESCRIPTION

Note: For most normal file system maintenance, the function of **ncheck** is subsumed by **fsck**(8).

ncheck generates a pathname versus i-number list of files for the indicated *filesystem*. Names of directory files are followed by '**.**'

The report is in no useful order, and probably should be sorted.

OPTIONS

–i *numbers*
: Report only those files whose i-*numbers* follow.

–a
: Print the names '**.**' and '**..**', which are ordinarily suppressed.

–s
: Report only special files and files with set-user-ID mode. This is intended to discover concealed violations of security policy.

SEE ALSO

sort(1V), **dcheck**(8), **fsck**(8), **icheck**(8)

DIAGNOSTICS

When the filesystem structure is improper, '**??**' denotes the "parent" of a parentless file and a pathname beginning with '**...**' denotes a loop.

NAME

ndbootd – ND boot block server

SYNOPSIS

ndbootd [**–dv**]

DESCRIPTION

ndbootd sends boot blocks to diskless Sun-2 system clients that request them using the (now obsolete) ND protocol. This server uses the boot block contained in the file **/tftpboot/sun2.bb**. A client must appear in the **ethers**(5) and **hosts**(5) databases, in order for the request to be served. In determining whether to serve the client, **ndbootd** checks the **/tftpboot** directory for a file whose name is the client's IP address in hexadecimal notation. For example, if the file **/tftpboot/C00901AD** exists, the machine at IP address 192.9.1.173 can be served. This file normally contains the boot program that is sent to the client by **tftpd**(8C).

Only root can invoke **ndbootd**.

OPTIONS

–d Debug. Display information about ignored packets, retransmissions, and address translation.

–v Verbose. Show a detailed listing of packets sent and received, etc.

If either option is used, all output is sent to the invoking terminal. Otherwise, error output (if any) appears on the console.

FILES

/tftpboot	bootfiles directory
/tftpboot/sun2.bb	boot blocks
/tftpboot/????????	boot programs for clients

SEE ALSO

ethers(5), **hosts**(5), **boot**(8S), **tftpd**(8C)

NAME

netconfig – PNP boot service

SYNOPSIS

/single/netconfig [**−e**] [**−n**]

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

netconfig is used both for automatic installation of new diskful systems, and during routine booting of all systems. The sequence of actions taken by **netconfig** depends on which of these situations is in effect, but it always sets the hostname, domainname, time, timezone, and interface IP address. If the system is newly installed on the network, it does more, perhaps interrogating the user about system configuration.

netconfig is invoked with the **−e** option from the **/etc/rc.boot** script.

Invoked without options, **netconfig** may perform PNP set up, including set up of files, passwords, and secure RPCs. Unless **−n** is specified, it writes **/etc/net.conf**, which is read later by **rc.boot**. This includes the **VERBOSE** flag, derived from **NVRAM** data, which controls the verbosity of the commands in **rc.boot**.

Routine Booting

Boot servers use information stored locally in Network Interface Service (NIS) acquiring it over the network, except that they get the time from the *timehost* system if it is up. The following describes the steps taken by boot clients: diskful clients, diskless clients, and network clients.

Boot clients first invoke **rarp** to acquire an IP address. This is followed by a **ICMP Netmask** request to obtain the IP subnetwork mask, and then a **PNP_WHOAMI** RPC to determine the system's name, NIS domain, and time zone. Then the systems clock is set using the RFC 868 time service. If **PNP_WHOAMI** fails, a **PNP_SETUP** sequence is followed by set up of **/etc/passwd** and other files.

OPTIONS

−e　Check shell environment variables. This option is specified during routine boot. **HOSTNAME** and **DOMAINNAME** are used to determine if the system is an NIS server using local NIS maps. Otherwise, if **NETWORKED** is **YES**, **netconfig** probes the network for network configuration. **MUST_SETUP** requires writing **/etc/passwd** and other files for setup in restricted network environments.

−n　Used in conjunction with '**−e**', this does not probe the network for anything but just sets the hostname and domainname of the system from the environment variables **HOSTNAME** and **DOMAINNAME** respectively. Does not write the **/etc/net.conf** file.

FILES

/var/yp/*domainname***/netmasks**
/var/yp/*domainname***/hosts**

SEE ALSO

pnp(3R), **pnpboot**(8C), **pnpd**(8C), **rarpd**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

netstat – show network status

SYNOPSIS

netstat [**–aAn**] [**–f** *address_family*] [**system**] [**core**]

netstat [**–n**] [**–s**] [**–m** | **–i** | **–r**] [**–f** *address_family*] [**system**] [**core**]

netstat [**–n**] [**–I** *interface*] *interval* [**system**] [**core**]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

netstat displays the contents of various network-related data structures in various formats, depending on the options you select.

The first form of the command displays a list of active sockets for each protocol. The second form selects one from among various other network data structures. The third form displays running statistics of packet traffic on configured network interfaces; the *interval* argument indicates the number of seconds in which to gather statistics between displays.

The default value for the **system** argument is **/vmunix**; for *core*, the default is **/dev/kmem**.

OPTIONS

–a Show the state of all sockets; normally sockets used by server processes are not shown.

–A Show the address of any protocol control blocks associated with sockets; used for debugging.

–f *address_family*
Limit statistics or address control block reports to those of the specified *address_family*, which can be one of:

inet For the AF_INET address family, or
unix For the AF_UNIX family.

–i Show the state of interfaces that have been auto-configured. Interfaces that are statically configured into a system, but not located at boot time, are not shown.

–I *interface* Highlight information about the indicated *interface* in a separate column; the default (for the third form of the command) is the interface with the most traffic since the system was last rebooted. *interface* can be any valid interface listed in the system configuration file, such as **ie0** or **le0**.

–m Show the statistics recorded by management routines for the network's private buffer pool.

–n Show network addresses as numbers. **netstat** normally displays addresses as symbols. This option may be used with any of the display formats.

–r Show the routing tables. (When –s is also present, show routing statistics instead.)

–s Show per-protocol statistics. When used with the **–r** option, show routing statistics.

–t Replace queue length information with timer information.

DISPLAYS

Active Sockets (First Form)

The display for each active socket shows the local and remote address, the send and receive queue sizes (in bytes), the protocol, and the internal state of the protocol.

The symbolic format normally used to display socket addresses is either:

hostname.port

when the name of the host is specified, or:

network.port

if a socket address specifies a network but no specific host. Each *hostname* and *network* is shown according to its entry in the **/etc/hosts** or the **/etc/networks** file, as appropriate.

If the network or hostname for an address is not known (or if the **–n** option is specified), the numerical network address is shown. Unspecified, or ''wildcard'', addresses and ports appear as ''*''. (For more information regarding the Internet naming conventions, refer to **inet**(3N)).

TCP Sockets

The possible state values for TCP sockets are as follows:

CLOSED	Closed: the socket is not being used.
LISTEN	Listening for incoming connections.
SYN_SENT	Actively trying to establish connection.
SYN_RECEIVED	Initial synchronization of the connection under way.
ESTABLISHED	Connection has been established.
CLOSE_WAIT	Remote shut down: waiting for the socket to close.
FIN_WAIT_1	Socket closed, shutting down connection.
CLOSING	Closed, then remote shutdown: awaiting acknowledgement.
LAST_ACK	Remote shut down, then closed: awaiting acknowledgement.
FIN_WAIT_2	Socket closed, waiting for shutdown from remote.
TIME_WAIT	Wait after close for remote shutdown retransmission.

Network Data Structures (Second Form)

The form of the display depends upon which of the **–m**, **–i**, **–h** or **–r**, options you select. (If you specify more than one of these options, **netstat** selects one in the order listed here.)

Routing Table Display

The routing table display lists the available routes and the status of each. Each route consists of a destination host or network, and a gateway to use in forwarding packets. The *flags* column shows the status of the route (**U** if ''up''), whether the route is to a gateway (**G**), and whether the route was created dynamically by a redirect (**D**).

Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface.

The **refcnt** column gives the current number of active uses per route. (Connection-oriented protocols normally hold on to a single route for the duration of a connection, whereas connectionless protocols obtain a route while sending to the same destination.)

The **use** column displays the number of packets sent per route.

The *interface* entry indicates the network interface utilized for the route.

Cumulative Traffic Statistics (Third Form)

When the *interval* argument is given, **netstat** displays a table of cumulative statistics regarding packets transferred, errors and collisions, the network addresses for the interface, and the maximum transmission unit (''mtu''). The first line of data displayed, and every 24th line thereafter, contains cumulative statistics from the time the system was last rebooted. Each subsequent line shows incremental statistics for the *interval* (specified on the command line) since the previous display.

SEE ALSO

hosts(5), **networks**(5), **protocols**(5), **services**(5) **iostat**(8), **trpt**(8C), **vmstat**(8)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

The kernel's tables can change while **netstat** is examining them, creating incorrect or partial displays.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

newaliases rebuilds the random access data base for the mail aliases file **/etc/aliases**. It is run automatically by **sendmail**(8) (in the default configuration) whenever a message is sent.

FILES

/etc/aliases

SEE ALSO

aliases(5), **sendmail**(8)

NAME

newfs – create a new file system

SYNOPSIS

/usr/etc/newfs [**–Nv**] [*mkfs-options*] *raw-special-device*

DESCRIPTION

newfs is a "friendly" front-end to the **mkfs**(8) program. On Sun systems, the disk type is determined by reading the disk label for the specified *raw-special-device*.

raw-special-device is the name of a raw special device residing in **/dev**, including the disk partition, where you want the new file system to be created. If you want to make a file system on **sd0[a-h]**, specify **sd0[a-h]**, **rsd0[a-h]** or **/dev/rsd0[a-h]**; if you only specify **sd0[a-h]**, **newfs** will find the proper device.

newfs then calculates the appropriate parameters to use in calling **mkfs**, and builds the file system by forking **mkfs**.

You must be super-user to use this command.

OPTIONS

–N Print out the file system parameters without actually creating the file system.

–v Verbose. **newfs** prints out its actions, including the parameters passed to **mkfs**.

mkfs-options

Options that override the default parameters passed to **mkfs**(8) are:

–a *apc* Number of alternates per cylinder (SCSI devices only).

–b *block-size*

The block size of the file system in bytes. The default is 8192.

–c *#cylinders/group*

The number of cylinders per cylinder group in a file system. The default is 16.

–d *rotdelay*

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–f *frag-size*

The fragment size of the file system in bytes. The default is 1024.

–i *bytes/inode*

This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

–m *free-space%*

The percentage of space reserved from normal users; the minimum free space threshold. The default is 10%.

–o *optimization*

(**space** or **time**). The file system can either be instructed to try to minimize the time spent allocating blocks, or to try to minimize the space fragmentation on the disk. If the minimum free space threshold (as specified by the **–m** option) is less than 10%, the default is to optimize for **space**; if the minimum free space threshold is greater than or equal to 10%, the default is to optimize for **time**.

–r *revolutions/minute*

The speed of the disk in revolutions per minute (normally 3600).

–s *size* The size of the file system in sectors.

−t *#tracks/cylinder*
The number of tracks per cylinders on the disk. The default is 16.

−n *#rotational-positions*
The number of distinguished rotational positions. The default is 8.

EXAMPLES

The following example verbosely displays the parameters for the raw special device, **sd0a**, but does not actually create a new file system:

```
example% /usr/etc/newfs −vN sd0a
mkfs −N /dev/rsd0a 16048 34 8 8192 1024 16 10 60 2048 t 0 -1
/dev/rsd0a:     16048 sectors in 59 cylinders of 8 tracks, 34 sectors
        8.2Mb in 4 cyl groups (16 c/g, 2.23Mb/g, 896 i/g)
super-block backups (for fsck −b#) at:
 32, 4432, 8832, 13232,
example%
```

SEE ALSO

fs(5), **fsck**(8), **installboot**(8S), **mkfs**(8), **tunefs**(8)

System and Network Administration

DIAGNOSTICS

newfs: *special* **No such file or directory**
The device specified does not exist, or a disk partition was not specified.

special: **cannot open**
You must be super-user to use this command.

NOTES

To install the bootstrap programs for a root partition, run **installboot**(8S) after **newfs**.

NAME

newkey – create a new key in the publickey database

SYNOPSIS

newkey **–h** *hostname*
newkey **–u** *username*

DESCRIPTION

newkey is normally run by the network administrator on the Network Interface Service (NIS) master machine in order to establish public keys for users and super-users on the network. These keys are needed for using secure RPC or secure NFS.

newkey will prompt for the login password of the given username and then create a new public/secret key pair in **/etc/publickey** encrypted with the login password of the given user.

Use of this program is not required: users may create their own keys using **chkey**(1).

OPTIONS

–h *hostname* Create a new public key for the super-user at the given hostname. Prompts for the root password of the given hostname.

–u *username* Create a new public key for the given username. Prompts for the NIS password of the given username.

SEE ALSO

chkey(1), **keylogin**(1), **publickey**(5), **keyserv**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

nfsd, biod – NFS daemons

SYNOPSIS

/usr/etc/nfsd [*nservers*]

/usr/etc/biod [*nservers*]

DESCRIPTION

nfsd starts the daemons that handle client filesystem requests. *nservers* is the number of file system request daemons to start. This number should be based on the load expected on this server. Eight seems to be a good number.

biod starts *nservers* asynchronous block I/O daemons. This command is used on a NFS client to buffer cache handle read-ahead and write-behind. The magic number for *nservers* in here is also eight.

When a file that is opened by a client is unlinked (by the server), a file with a name of the form **.nfs***XXX* (where *XXX* is a number) is created by the client. When the open file is closed, the **.nfs***XXX* file is removed. If the client crashes before the file can be closed, the **.nfs***XXX* file is not removed.

FILES

.nfs*XXX*	client machine pointer to an open-but-unlinked file

SEE ALSO

exports(5), **mountd**(8C)

NAME

nfsstat – Network File System statistics

SYNOPSIS

nfsstat [−cmnrsz]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

nfsstat displays statistical information about the NFS (Network File System) and RPC (Remote Procedure Call), interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

nfsstat −cnrs

That is, display everything, but reinitialize nothing.

OPTIONS

−c Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **−n** and **−r** options to print client NFS or client RPC information only.

−m Display statistics for each NFS mounted file system. This includes the server name and address, mount flags, current read and write sizes, the retransmission count, and the timers used for dynamic retransmission.

−n Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **−c** and **−s** options to print client or server NFS information only.

−r Display RPC information.

−s Display server information.

−z Zero (reinitialize) statistics. This option is for use by the super-user only, and can be combined with any of the above options to zero particular sets of statistics after printing them.

DISPLAYS

The server RPC display includes the fields:

calls total number of RPC calls received

badcalls total number of calls rejected

nullrecv number of times no RPC packet was available when trying to receive

badlen number of packets that were too short

xdrcall number of packets that had a malformed header

The server NFS display shows the number of NFS calls received (**calls**) and rejected (**badcalls**), and the counts and percentages for the various calls that were made.

The client RPC display includes the following fields:

calls total number of RPC calls sent
badcalls total of calls rejected by a server
retrans number of times a call had to be retransmitted
badxid number of times a reply did not match the call
timeout number of times a call timed out
wait number of times a call had to wait on a busy **CLIENT** handle
newcred number of times authentication information had to be refreshed

The client NFS display shows the number of calls sent and rejected, as well as the number of times a **CLIENT** handle was received (**nclget**), the number of times a call had to sleep while awaiting a handle (**nclsleep**), as well as a count of the various calls and their respective percentages.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory

NAME

listen, nlsadmin – network listener service administration for RFS

SYNOPSIS

nlsadmin [**−mx**] [**−edr** *service_code net_spec*] [**−ikqsv** *net_spec*]
[**−lt** *addr net_spec*] [**−a** *service_code* [**−p** *modules*] **−c** *command* **−y** *comment net_spec*]
[**−qz** *code net_spec*] [**−z** *code net_spec*] [*net_spec*]

/usr/etc/listen

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

nlsadmin configures, initiates and terminates network listener (**listen**) servers for the local host. Each network (transport provider) has an associated **listen** daemon to service it locally. The **listen** daemon for each is configured separately. A **listen** daemon accepts network service requests when they arrive, and spawns servers in response to those requests. It can be used on any network (transport provider) that conforms to the transport provider specification.

nlsadmin can also report on the listener processes on a machine, either individually (per network) or collectively.

Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted.

nlsadmin without any options gives a brief usage message.

The *net_spec* argument to **nlsadmin** refers to a particular **listen** daemon. Specifically, *net_spec* is the relative path name of the entry under **/dev** for a given network.

−x Report the status of all of the listener processes installed on this machine.

−e *service_code net_spec*
−d *service_code net_spec*
Enable or disable, respectively, the service indicated by *service_code* for the specified network. The service must have previously been added to the listener for that network (see the **−a** option). When a listener is disabled, processes serving prior requests continue until they complete.

−r *service_code net_spec*
Remove the entry for the *service_code* from that listener's list of services.

−i *net_spec* Initialize or change a listener process for the network specified by *net_spec*. That is, create and initialize the files required by the listener. Initializing a listener with this option does not start it running. The listener must be initialized before assigning addressing or services. Note: the listener should only be initialized once for a given network.

−q *net_spec* Query the status of the listener process for the specified network. If the listener process is active, **nlsadmin** exits with a status of 0. If no such process is active, the exit code is 1. The exit code will be greater than 1 if there is an error.

−s *net_spec*
−k *net_spec*
Start or kill, respectively, the listener process for the indicated network. When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected. The listener runs under its own ID of **listen** with group ID (GID) **adm**. This GID appear in the system password file **/etc/passwd**; the **HOME** directory listed for the GID is concatenated with *net_spec* to determine the location of the listener configuration information for each network.

nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super-user.

−v *net_spec* Verbose. Report on the servers associated with *net_spec*, giving the service code, status, command, and comment for each.

−l *addr net_spec* Change or set the address for the general listener service. This is the address generally used by remote processes to access the servers available through the listener (see the **−a** option). *addr* is the transport address on which to listen, and is interpreted using a syntax that allows for a variety of address formats. By default *addr* is interpreted as the symbolic ASCII representation of the transport address. An *addr* preceded by a '\x' (BACKSLASH-X) lets you enter an address in hexadecimal notation. Note: *addr* must be quoted if it contains any blanks. If *addr* is just a dash ('—'), **nlsadmin** merely reports the currently configured address.

A change of address does not take effect until the next time the listener for that network is started.

−t *addr net_spec* Change or set the address on which the listener listens for requests for terminal service. Otherwise, this is similar to **−l**. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code **1** (see the **−a** option).

[**−m**] **−a** *service_code* **−c** *cmd* **−y** *comment net_spec*

Add a new service to the list of services available through the indicated listener. *service_code* is the code for the service, *cmd* is the command to be invoked in response to that service code, comprised of the full path name of the server and its arguments, and *comment* is a brief (free-form) description of the service for use in various reports. Note: *cmd* must be quoted if it contains arguments for the server. Similarly, *comment* must also be quoted, so as to appear to be a single word to the shell. When a service is added, it is initially enabled (see the **−e** and **−d** options).

If the **−m** option is specified, the entry is marked as an administrative entry. Service codes **1** through **100** are reserved for administrative entries, which are those that require special handling internally. In particular, code **1** is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

A service must explicitly be added to the listener for each network on which that service is to be available. This operation is normally performed only when the service is installed on a machine, or when populating the list of services for a new network.

−qz *code net_spec*

Query the status of the service with service code *code* on network *net_spec*, Exit with a status of 0 if the service is enabled, 1 if the service is disabled, or greater than 1 on error.

−z *code net_spec* Print a report on the server associated with *net_spec* that has service code *code*, giving the same information as in the **−v** option.

net_spec Print the status of the listener process for *net_spec*.

DIAGNOSTICS

If the command is not run under the proper ID, an error message is sent to the standard error, and the command terminates.

FILES

/usr/etc/listen
/usr/net/nls/*net_spec*

SEE ALSO

Network Programming

NAME

nslookup – query domain name servers interactively

SYNOPSIS

nslookup [**–l**] [*address*]

DESCRIPTION

nslookup is an interactive program to query Internet domain name servers. The user can contact servers to request information about a specific host or print a list of hosts in the domain.

OPTIONS

–l Use the local host's name server instead of the servers in **/etc/resolv.conf**. (If **/etc/resolv.conf** does not exist or does not contain server information, the **–l** option does not have any effect).

address Use the name server on the host machine with the given Internet address.

USAGE

Overview

The Internet domain name-space is tree-structured, with top-level domains such as:

COM	commercial establishments
EDU	educational institutions
GOV	government agencies
MIL	MILNET hosts

If you are looking for a specific host, you need to know something about the host's organization in order to determine the top-level domain it belongs to. For instance, if you want to find the Internet address of a machine at UCLA, do the following:

- Connect with the root server using the **root** command. The root server of the name space has knowledge of the top-level domains.
- Since UCLA is a university, its domain name is **ucla.edu**. Connect with a server for the **ucla.edu** domain with the command **server**ucla.edu. The response will print the names of hosts that act as servers for that domain. Note: the root server does not have information about **ucla.edu**, but knows the names and addresses of hosts that do. Once located by the root server, all future queries will be sent to the UCLA name server.
- To request information about a particular host in the domain (for instance, **locus**), just type the host name. To request a listing of hosts in the UCLA domain, use the **ls** command. The **ls** command requires a domain name (in this case, **ucla.edu**) as an argument.

Note: if you are connected with a name server that handles more than one domain, all lookups for host names must be fully specified with its domain. For instance, the domain **harvard.edu** is served by **seismo.css.gov**, which also services the **css.gov** and **cornell.edu** domains. A lookup request for the host **aiken** in the **harvard.edu** domain must be specified as **aiken.harvard.edu**. However, the

set *domain*= **name**

and

set defname

commands can be used to automatically append a domain name to each request.

After a successful lookup of a host, use the **finger** command to see who is on the system, or to finger a specific person. To get other information about the host, use the

set *querytype*= **value**

command to change the type of information desired and request another lookup. (**finger** requires the type to be A.)

Commands

Commands may be interrupted at any time by typing CTRL-C. To exit, type CTRL-D (EOF). The command line length must be less than 80 characters. Note: an unrecognized command will be interpreted as a host name.

host [*server*]
Look up information for *host* using the current default server or using *server* if it is specified.

server *domain*
lserver *domain*
Change the default server to *domain*. **lserver** uses the initial server to look up information about *domain* while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

root Changes the default server to the server for the root of the domain name space. Currently, the host **sri-nic.arpa** is used; this command is a synonym for '**lserver sri-nic.arpa**'.) The name of the root server can be changed with the **set root** command.

finger [*name* **]**
Connect with the finger server on the current host, which is defined by a previous successful lookup for a host's address information (see the **set** *querytype*= **A** command). As with the shell, output can be redirected to a named file using > and >>.

"**ls [−ah]**
List the information available for *domain*. The default output contains host names and their Internet addresses. The −**a** option lists aliases of hosts in the domain. The −**h** option lists CPU and operating system information for the domain. As with the shell, output can be redirected to a named file using > and >>. When output is directed to a file, hash marks are printed for every 50 records received from the server.

view*filename*
Sort and list the output of the **ls** command with **more**(1).

help

? Print a brief summary of commands.

set*keyword* [= *value*] This command is used to change state information that affects the lookups. Valid keywords are:

all Prints the current values of the various options to **set**. Information about the current default server and host is also printed.

[no]deb[ug]
Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer. The default is **nodebug**.

[no]def[name]
Append the default domain name to every lookup. The default is **nodefname.**

do[main]=*filename*
Change the default domain name to *filename*. The default domain name is appended to all lookup requests if **defname** option has been set. The default is the value in **/etc/resolv.conf**.

q[querytype]=*value*
Change the type of information returned from a query to one of:

A The host's Internet address (the default).
CNAME
The canonical name for an alias.
HINFO The host CPU and operating system type.
MD The mail destination.

MX The mail exchanger.
MB The mailbox domain name.
MG The mail group member.
MINFO The mailbox or mail list information.

(Other types specified in the RFC883 document are valid, but are not very useful.)

[no]recurse
Tell the name server to query other servers if it does not have the information. The default is **recurse**.

ret[ry]=*count*
Set the number of times to retry a request before giving up to *count*. When a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The default is *count* is **2**.

ro[ot]=*host*
Change the name of the root server to *host*. This affects the **root** command. The default root server is **sri-nic.arpa**.

t[timeout]=*interval*
Change the time-out for a reply to *interval* seconds. The default *interval is* **10** seconds.

[no]v[c]
Always use a virtual circuit when sending requests to the server. The default is **novc**.

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

Time-out
The server did not respond to a request after a certain amount of time (changed with **set** *timeout*= **value**) and a certain number of retries (changed with **set** *retry*= **value**).

No information
Depending on the query type set with the **set querytype** command, no information about the host was available, though the host name is valid.

Non-existent domain
The host or domain name does not exist.

Connection refused
Network is unreachable
The connection to the name or finger server could not be made at the current time. This error commonly occurs with **finger** requests.

Server failure
The name server found an internal inconsistency in its database and could not return a valid answer.

Refused
The name server refused to service the request.

The following error should not occur and it indicates a bug in the program.

Format error
The name server found that the request packet was not in the proper format.

FILES

/etc/resolv.conf initial domain name and name server addresses.

SEE ALSO

resolver(3), **resolv.conf**(5), **named**(8C)

RFC 1034, RFC 1035

System and Network Administration

NAME

nsquery – RFS name server query

SYNOPSIS

nsquery [**–h**] [*name*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

nsquery provides information about resources available to the host from both the local domain and from other domains. All resources are reported, regardless of whether the host is authorized to access them. When used with no options, **nsquery** identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is one of:

nodename	The report will include only those resources available from *nodename*.
domain.	The report will include only those resources available from *domain*.
domain.nodename	The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter '.', it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter '.', it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (*domain.resource*), the read/write permissions, the server (*nodename.domain*) that advertised the resource, and a brief textual description.

A remote domain must be listed in your **rfmaster** file in order to query that domain.

If no entries are found when **nsquery** is executed, the report header is printed.

If your host cannot contact the domain name server, an error message will be sent to standard error.

OPTIONS

–h Do not print header.

EXAMPLE

The following example displays the resources available from the domain **sunrfs**:

```
example% nsquery sunrfs.
RESOURCE        ACCESS        SERVER          DESCRIPTION

LOCAL_SUN3      read-only     sunrfs.estale
LOCAL_SUN4      read-only     sunrfs.estale
LOCAL_SHARE     read-only     sunrfs.estale
```

SEE ALSO

rfmaster(5), **adv**(8), **unadv**(8)

NAME

old-analyze, analyze – postmortem system crash analyzer

SYNOPSIS

/usr/old/analyze [**–dfmvD**] [**–s** *swapfile*] *corefile* [**system**]

DESCRIPTION

analyze is the post-mortem analyzer for the state of the paging system. In order to use **analyze** you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see **panic**(8S)).

The **analyze** program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file) to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw I/O is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to **analyze**:

–d Print the (sorted) paging area usage.

–f Dump the free list.

–m Dump the entire coremap state.

–v (Long unused.) Use a hugely verbose output format.

–D Print the diskmap for each process.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a **pstat**(8) to weed out such processes.

It is possible to look at the core dump with **adb**(1) if you do

adb –k /vmunix /vmcore

FILES

/vmunix default system namelist

SEE ALSO

adb(1), **ps**(1), **panic**(8S), **pstat**(8)

DIAGNOSTICS

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

NAME

pac – printer/plotter accounting information

SYNOPSIS

/usr/etc/pac [**–cmrs**] [**–P***printer*] [**–p***price*] [*username*...]

DESCRIPTION

pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. The accounting file is taken from the **af** field of the **printcap** entry for the printer. If any *username*s are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

OPTIONS

–c Sort the output by cost; usually the output is sorted alphabetically by name.

–m Disregard machine names. Normally, print jobs submitted by a user from different machines would be counted separately for each machine.

–r Reverse the sorting order.

–s Summarize the accounting information on the summary accounting file. The name of the summary file is the name of the accounting file with '**_sum**' appended to it.

–P*printer* Do accounting for the named *printer*. If this option is not used, the printer specified by the PRINTER environment variable will be used if it is present; otherwise accounting is done for the default printer.

–p*price* Use the value *price* for the cost in dollars per page/foot instead of the default value of 0.02.

FILES

/etc/printcap

SEE ALSO

printcap(5)

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

panic – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily, it displays a message of the form

> **panic:** *why i gave up the ghost*

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in **reboot**(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure, the system will then resume multiuser operations.

The system has a large number of internal consistency checks; if one of these fails, it will panic with a very short message indicating which one failed.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, you can run the program **savecore**(8) to preserve a copy of this core image and kernel namelist for later perusal. See **savecore**(8) for details.

To analyze a dump you should begin by running **adb**(1) with the **–k** flag on the core dump, as described in *Debugging Tools.*

The most common cause of system failures is hardware failure, which can reflect itself in different ways.

See DIAGNOSTICS for some messages that you may encounter, with some hints as to causes. In each case there is a possibility that a hardware or software error produced the message in some unexpected way.

FILES

/vmunix	the system kernel
/etc/rc.local	script run when the local system starts up

SEE ALSO

adb(1), **old-analyze**(8), **reboot**(8) **sa**(8), **savecore**(8)

Debugging Tools

DIAGNOSTICS

IO err in push

hard IO err in swap The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow

This really should not be a panic, but until the data structure is fixed, involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger by changing the value of **ncallout** in the **param.c** file, and then rebuild your system.

trap type *type*, **pid** *process-id*, **pc =** *program-counter*, **sr =** *status-register*, **context** *context-number*

A unexpected trap has occurred within the system; typical trap types are:

- Bus error
- Address error
- Illegal instruction
- Divide by zero
- Chk instruction
- Trapv instruction
- Privilege violation
- Trace
- 1010 emulator trap
- 1111 emulator trap
- Stack format error

- Uninitialized interrupt
- Spurious interrupt

The favorite trap types in system crashes are ''Bus error'' or ''Address error'', indicating a wild reference. The *process-id* is the ID of the process running at the time of the fault, *program-counter* is the hexadecimal value of the program counter, *status-register* is the hexadecimal value of the status register, and *context-number* is the context that the process was running in. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/usr/etc/ping *host* [*timeout*]

/usr/etc/ping [–s] [**–lrRv**] *host* [*packetsize*] [*count*]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

ping utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from the specified *host* , or network gateway. ECHO_REQUEST datagrams, or "pings," have an IP and ICMP header, followed by a *struct*timeval, and then an arbitrary number of bytes to pad out the packet. If *host* responds, **ping** will print *host* **is alive** on the standard output and exit. Otherwise after *timeout* seconds, it will write **no answer from** *host*. The default value of *timeout* is 20 seconds.

When the **–s** flag is specified, **ping** sends one datagram per second, and prints one line of output for every ECHO_RESPONSE that it receives. No output is produced if there is no response. In this second form, **ping** computes round trip times and packet loss statistics; it displays a summary of this information upon termination or timeout. The default datagram packet size is 64 bytes, or you can specify a size with the *packetsize* command-line argument. If an optional *count* is given, **ping** sends only that number of requests.

When using **ping** for fault isolation, first '**ping**' the local host to verify that the local network interface is running.

OPTIONS

–l Loose source route. Use this option in the IP header to send the packet to the given host and back again. Usually specified with the **–R** option.

–r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to **ping** a local host through an interface that has been dropped by the router daemon, see **routed**(8C).

–R Record route. Sets the IP record route option, which will store the route of the packet inside the IP header. The contents of the record route will only be printed if the **–v** option is given, and only be set on return packets if the target host preserves the record route option across echos, or the **–l** option is given.

–v Verbose output. List any ICMP packets, other than ECHO_RESPONSE, that are received.

SEE ALSO

icmp(4P), **ifconfig**(8C), **netstat**(8C), **rpcinfo**(8C), **spray**(8C)

NAME

pnpboot, pnp.s386 – pnp diskless boot service

SYNOPSIS

/tftpboot/pnp.s386

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

pnp.s386 is a level 2 boot program that requests actions necessary to set up a diskless workstation on the network.

The PNP diskless boot service is used by diskless workstations at installation time to locate a server that will configure the diskless client.

The last steps of the level 1 boot (from the PROM) are to load the level 2 program through **rarpd**(8C) and **tftpd**(8C). The first step in the boot sequence is RARP to acquire an IP address. This is followed by TFTP service calls to acquire the **pnp.sun*** program file needed for the client's architecture. A **PNP_ACQUIRE** RPC is then broadcast to locate a server willing to configure the diskless client.

A **PNP_SETUP** is issued to the server which returns one of three statuses: success, failure, or in_progress. As long as the server responds with a status of in_progress the client will periodically issue a **PNP_POLL** until the status changes to either success or failure.

The last step is to reboot the client. This goes through a RARP, TFTP, BOOT sequence, with the boot using the normal **boot.sun*** file and **bootparamd**(8) service.

The system will have been set up using the IP address returned in the first step and a system name will have been assigned.

FILES

/tftpboot/pnp.sun*

SEE ALSO

bootparam(3R), **bootparams**(5) **boot**(8S), **bootparamd**(8), **ipallocd**(8C), **netconfig**(8C), **pnpd**(8C), **rarpd**(8C), **tftpd**(8C)

NAME

pnpd – PNP daemon

SYNOPSIS

/usr/etc/rpc.pnpd

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

pnpd is used during routine booting of systems to determine their network configuration, and by new systems to configure themselves on a network. **pnpd** adds and removes diskless clients of the boot server on which it is running. The **pnpd** daemon is normally invoked in **rc.local**. The RPCs are used by **netconfig**(8C), **pnp.s386** (see **pnpboot**(8C)), and **client**(8).

The **bootservers** Network Interface Service (NIS) map specifies limits on server capacity and default swap size.

FILES

/export/exec/*arch*
  symbolic link to **/export/exec/***arch.release*

/export/exec/*arch.release*
  symbolic link to **/usr** for the architecture

/export/exec/*arch.release***/boot**
  root binaries

SEE ALSO

pnp(3R), **client**(8), **ipallocd**(8C), **netconfig**(8C), **pnpboot**(8C)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

portmap – TCP/IP port to RPC program number mapper

SYNOPSIS

/usr/etc/portmap

DESCRIPTION

portmap is a server that converts TCP/IP protocol port numbers into RPC program numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell **portmap** what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact **portmap** on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by **inetd**(8C), so **portmap** must be started before **inetd** is invoked.

SEE ALSO

inetd.conf(5), **inetd**(8C), **rpcinfo**(8C)

BUGS

If **portmap** crashes, all servers must be restarted.

NAME

praudit – print contents of an audit trail file

SYNOPSIS

praudit [**–lrs**] [**–d***del*] [*filename* ...]

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

praudit reads the listed *filename*s (or standard input, if no *filename* is specified) and interprets the data as audit trail records as defined in **audit_control**(5). By default, times, security labels, user and group IDs (UIDs and GIDs, respectively) are converted to their ASCII representation. Record type and event fields are converted to long ASCII representation. A maximum of 100 audit files can be specified on the command line.

OPTIONS

–l Print records one line per record. The record type and event fields are always converted to their short ASCII representation.

–r Print records in their raw form. Times, security labels, UIDs, GIDs, record types, and events are displayed as integers. Currently, labels are not used and are displayed as zero in this mode. This option and the **–s** option are exclusive. If both are used, a format usage error message is output.

–s Print records in their short form. All numeric fields are converted to ASCII and displayed. The short ASCII representations for the record type and event fields are used. Security labels are displayed in their short representation. Again, labels are not currently used. This option and the **–r** option are exclusive. If both are used, a format usage error message is output.

–d*del* Use *del* as the field delimiter instead of the default delimiter, which is the comma. If *del* has special meaning for the shell, it must be quoted. The maximum size of a delimiter is four characters.

FILES

/etc/passwd

SEE ALSO

audit(2), **setuseraudit**(2), **getauditflags**(3), **audit_control**(5)

NAME

pstat – print system facts

SYNOPSIS

/usr/etc/pstat [**–afipSsT**] [**–u** *pid*] [*system* [*corefile*]]

DESCRIPTION

pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in **/dev/kmem**. The required namelist is taken from **/vmunix** unless *system* is specified.

OPTIONS

–a Under **–p**, describe all process slots rather than just active ones.

–f Print the open file table with these headings:

LOC The memory address of this table entry.
TYPE The type of object the file table entry points to.
FLG Miscellaneous state variables encoded thus:
- R open for reading
- W open for writing
- A open for appending
- S shared lock present
- X exclusive lock present
- I signal pgrp when data ready

CNT Number of processes that know this open file.
MSG Number of references from message queue.
DATA The location of the vnode table entry or socket for this file.
OFFSET The file offset (see **lseek**(2V)).

–i Print the inode table including the associated vnode entries with these headings:

ILOC The memory address of this table entry.
IFLAG Miscellaneous inode state variables encoded thus:
- A inode access time must be corrected
- C inode change time must be corrected
- L inode is locked
- R inode is being referenced
- U update time (**fs**(5)) must be corrected
- W wanted by another process (L flag is on)

IDEVICE Major and minor device number of file system in which this inode resides.
INO I-number within the device.
MODE Mode bits in octal, see **chmod**(2V).
NLK Number of links to this inode.
UID User ID of owner.
SIZE/DEV Number of bytes in an ordinary file, or major and minor device of special file.
VFLAG Miscellaneous vnode state variables encoded thus:
- R root of its file system
- S shared lock applied
- E exclusive lock applied
- Z process is waiting for a shared or exclusive lock

CNT Number of open file table entries for this vnode.
SHC Reference count of shared locks on the vnode.
EXC Reference count of exclusive locks on the vnode (this may be '> **1**' if, for example, a file descriptor is inherited across a fork).
TYPE Vnode file type, either VNON (no type), VREG (regular), VDIR (directory), VBLK (block device), VCHR (character device), VLNK (symbolic link), VSOCK (socket), VFIFO (named pipe), or VBAD (bad).

−p Print process table for active processes with these headings:

LOC	The memory address of this table entry.
S	Run state encoded thus: 0 no process 1 awaiting an event 2 (abandoned state) 3 runnable 4 being created 5 being terminated 6 stopped (by signal or under trace)
F	Miscellaneous state variables, ORed together (hexadecimal): 0000001 loaded 0000002 a system process (scheduler or page-out daemon) 0000004 locked for swap out 0000008 swapped out during process creation 0000010 process is being traced 0000020 tracing parent has been told that process is stopped 0000040 user settable lock in memory 0000080 in page-wait 0000100 prevented from swapping during **fork**(2V) 0000200 will restore old mask after taking signal 0000400 exiting 0000800 doing physical I/O 0001000 process resulted from a **vfork**(2) which is not yet complete 0002000 another flag for **vfork**(2) 0004000 process has no virtual memory, as it is a parent in the context of **vfork**(2) 0008000 process is demand paging pages from its executable image vnode 0010000 process has advised of sequential VM behavior with **vadvise**(2) 0020000 process has advised of random VM behavior with **vadvise**(2) 0080000 process is a session process group leader 0100000 process is tracing another process 0200000 process needs a profiling tick 0400000 process is scanning descriptors during select 4000000 process has done record locks 8000000 process is having its system calls traced
PRI	Scheduling priority, see **getpriority**(2).
SIG	Signals received (signals 1-32 coded in bits 0-31).
UID	Real user ID.
SLP	Amount of time process has been blocked.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level, see **getpriority**(2).
PGRP	Process number of root of process group.
PID	The process ID number.
PPID	The process ID of parent process.
RSS	Resident set size — the number of physical page frames allocated to this process.
SRSS	RSS at last swap (0 if never swapped).

SIZE	The size of the process image. That is, the sum of the data and stack segment sizes, not including the sizes of any shared libraries.
WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.

−S Print the streams table with these headings:

LOC	The memory address of this table entry.
WRQ	The address of this stream's write queue.
VNODE	The address of this stream's vnode.
DEVICE	Major and minor device number of device to which this stream refers.
PGRP	This stream's process group number.
SIGIO	The process id or process group that has this stream **open**( ).
FLG	Miscellaneous stream state variables encoded thus:

I	waiting for **ioctl**( ) to finish
R	read/recvmsg is blocked
W	write/putmsg is blocked
P	priority message is at stream head
H	device has been ''hung up'' (**M_HANGUP**)
O	waiting for open to finish
M	stream is linked under multiplexor
D	stream is in message-discard mode
N	stream is in message-nondiscard mode
E	fatal error has occurred (**M_ERROR**)
T	waiting for queue to drain when closing
2	waiting for previous **ioctl**( ) to finish before starting new one
3	waiting for acknowledgment for **ioctl**( )
B	stream is in non-blocking mode
A	stream is in asynchronous mode
o	stream uses old-style no-delay mode
S	stream has had **TOSTOP** set
C	VTIME clock running
V	VTIME timer expired
r	collision on **select**( ) for reading
w	collision on **select**( ) for writing
e	collision on **select**( ) for exceptional condition

The queues on the write and read sides of the stream are listed for each stream. Each queue is printed with these headings:

NAME	The name of the module or driver for this queue.
COUNT	The approximate number of bytes on this queue.
FLG	Miscellaneous state variables encoded thus:

E	queue is enabled to run
R	someone wants to get from this queue when it becomes non-empty
W	someone wants to put on this queue when it drains
F	queue is full
N	queue should not be enabled automatically by a putq

MINPS	The minimum packet size for this queue.
MAXPS	The maximum packet size for this queue, or **INF** if there is no maximum.
HIWAT	The high-water mark for this queue.
LOWAT	The low-water mark for this queue.

−s Print information about swap space usage:

allocated:	The amount of swap space (in bytes) allocated to private pages.
reserved:	The number of swap space bytes not currently allocated, but claimed by memory mappings that have not yet created private pages.
used:	The total amount of swap space, in bytes, that is either allocated or reserved.
available:	The total swap space, in bytes, that is currently available for future reservation and allocation.

−T Print the number of used and free slots in the several system tables. This is useful for checking to see how full system tables have become if the system is under heavy load. Shows both used and cached inodes.

−u *pid* Print information about the process with ID *pid*.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

ps(1), **chmod**(2V), **fork**(2V), **getpriority**(2), **lseek**(2V), **stat**(2V), **vadvise**(2), **vfork**(2), **fs**(5) **iostat**(8), **vmstat**(8)

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **−T**; even more useful if these tables were dynamically allocated.

NAME

pwck – check password database entries

SYNOPSIS

/usr/etc/pwck [*filename*]

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

pwck checks that a file in **passwd**(5) does not contain any errors; it checks the **/etc/passwd** file by default.

FILES

/etc/passwd

DIAGNOSTICS

Too many/few fields
An entry in the password file does not have the proper number of fields.

No login name
The login name field of an entry is empty.

Bad character(s) in login name
The login name in an entry contains characters other than lower-case letters and digits.

First char in login name not lower case alpha
The login name in an entry does not begin with a lower-case letter.

Login name too long
The login name in an entry has more than 8 characters.

Invalid UID
The user ID field in an entry is not numeric or is greater than 65535.

Invalid GID
The group ID field in an entry is not numeric or is greater than 65535.

No login directory
The login directory field in an entry is empty.

Login directory not found
The login directory field in an entry refers to a directory that does not exist.

Optional shell file not found.
The login shell field in an entry refers to a program or shell script that does not exist.

No netgroup name
The entry is a Network Interface Service (NIS) entry referring to a netgroup, but no netgroup is present.

Bad character(s) in netgroup name
The netgroup name in an NIS entry contains characters other than lower-case letters and digits.

First char in netgroup name not lower case alpha
The netgroup name in an NIS entry does not begin with a lower-case letter.

SEE ALSO

group(5), **passwd**(5)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

pwdauthd – server for authenticating passwords

SYNOPSIS

/usr/etc/rpc.pwdauthd

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

pwdauthd is a server that determines authentication for users and groups. It handles authentication requests from **pwdauth**(3) and **grpauth**( ). Communication to and from **pwdauthd** is by means of RPC calls. The server is passed a *filename* and a *password*. It returns an integer value that specifies whether the *password* is valid. The possible return values are **PWA_VALID** if the name is valid, **PWA_INVALID** if the name is invalid, and **PWA_UNKNOWN** if validity cannot be determined because no adjunct files are present.

If **pwdauthd** is serving *pwdauth*, it determines whether the **passwd.adjunct** file exists. If not, it returns **PWA_UNKNOWN**. In this case, *pwdauth* knows to check the **/etc/passwd** file. Otherwise, the server calls **getpwanam**( ) (see **getpwaent**(3)) to get the entry for *filename* in either the local or the Network Interface Service (NIS) file for **passwd.adjunct**. If the encrypted password guess matches the encrypted password from the file, **pwdauthd** returns **PWA_VALID**. If the passwords do not match, it returns **PWA_INVALID**.

If **pwdauthd** is serving **grpauth**( ), it determines whether the **group.adjunct** file exists. If not, it returns **PWA_UNKNOWN**. In this case, **grpauth**( ) knows to check the **/etc/group** file. Otherwise, the server calls **getgranam**( ) (see **getgraent**(3)) to get the entry for *filename* in either the local or the NIS file for **group.adjunct**. If the encrypted password guess matches the encrypted password from the file, **pwdauthd** returns **PWA_VALID**. If the passwords do not match, it returns **PWA_INVALID**.

SEE ALSO

getgraent(3), **getpwaent**(3), **pwdauth**(3)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

quot – summarize file system ownership

SYNOPSIS

/usr/etc/quot [**–acfhnv**] [*filesystem*]

DESCRIPTION

quot displays the number of blocks (1024 bytes) in the named *filesystem* currently owned by each user.

OPTIONS

–a Generate a report for all mounted file systems.

–c Display three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

–f Display count of number of files as well as space owned by each user.

–h Estimate the number of blocks in the file — this doesn't account for files with holes in them.

–n Run the pipeline **ncheck filesystem | sort +0n | quot –n filesystem** to produce a list of all files and their owners.

–v Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

FILES

/etc/mtab	mounted file systems
/etc/passwd	to get user names

SEE ALSO

du(1V), **ls**(1V)

NAME

quotacheck – file system quota consistency checker

SYNOPSIS

/usr/etc/quotacheck [**–v**] [**–p**] *filesystem*...

/usr/etc/quotacheck [**–apv**]

DESCRIPTION

quotacheck examines each file system, builds a table of current disk usage, and compares this table against that stored in the disk quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

quotacheck expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, **quotacheck** will ignore the file system.

quotacheck is normally run at boot time from the **/etc/rc.local** file, see **rc**(8), before enabling disk quotas with **quotaon**(8).

quotacheck accesses the raw device in calculating the actual disk usage for each user. Thus, the file systems checked should be quiescent while **quotacheck** is running.

OPTIONS

–v Indicate the calculated disk quotas for each user on a particular file system. **quotacheck** normally reports only those quotas modified.

–a Check all the file systems indicated in **/etc/fstab** to be read-write with disk quotas.

–p Run parallel passes on the required file systems, using the pass numbers in **/etc/fstab** in an identical fashion to **fsck**(8).

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quotactl(2), **quotaon**(8), **rc**(8)

NAME

quotaon, quotaoff – turn file system quotas on and off

SYNOPSIS

/usr/etc/quotaon [**−v**] *filesystem*...
/usr/etc/quotaon [**−av**]

/usr/etc/quotaoff [**−v**] *filesystem*...
/usr/etc/quotaoff [**−av**]

DESCRIPTION

quotaon

quotaon announces to the system that disk quotas should be enabled on one or more file systems. The file systems specified must be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*.

quotaoff

quotaoff announces to the system that file systems specified should have any disk quotas turned off.

OPTIONS

quotaon

−a All file systems in **/etc/fstab** marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

−v Display a message for each file system where quotas are turned on.

quotaoff

−a Force all file systems in **/etc/fstab** to have their quotas disabled.

−v Display a message for each file system affected.

These commands update the status field of devices located in **/etc/mtab** to indicate when quotas are on or off for each file system.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quotactl(2), **fstab**(5), **mtab**(5)

NAME

rarpd – TCP/IP Reverse Address Resolution Protocol server

SYNOPSIS

/usr/etc/rarpd *interface* [*hostname*]

/usr/etc/rarpd –a

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rarpd starts a daemon that responds to Reverse Address Resolution Protocol (RARP) requests. The daemon forks a copy of itself that runs in background. It must be run as root.

RARP is used by machines at boot time to discover their Internet Protocol (IP) address. The booting machine provides its Ethernet Address in an RARP request message. Using the "ethers" and "hosts" databases, **rarpd** maps this Ethernet Address into the corresponding IP address which it returns to the booting machine in an RARP reply message. The booting machine must be listed in both databases for **rarpd** to locate its IP address. **rarpd** issues no reply when it fails to locate an IP address. The "ethers" and "hosts" databases may be contained either in files under **/etc** or in Network Interface Service (NIS) maps.

In the first synopsis, the *interface* parameter names the network interface upon which **rarpd** is to listen for requests. The *interface* parameter takes the "name unit" form used by **ifconfig**(8C). The second argument, *hostname*, is used to obtain the IP address of that interface. An IP address in "decimal dot" notation may be used for *hostname*. If *hostname* is omitted, the address of the interface will be obtained from the kernel. When the first form of the command is used, **rarpd** must be run separately for each interface on which RARP service is to be supported. A machine that is a router may invoke **rarpd** multiple times, for example:

/usr/etc/rarpd ie0 host
/usr/etc/rarpd ie1 host-backbone

In the second synopsis, **rarpd** locates all of the network interfaces present on the system and starts a daemon process for each one that supports RARP.

FILES

/etc/ethers
/etc/hosts

SEE ALSO

ethers(5), **hosts**(5), **policies**(5), **boot**(8S), **ifconfig**(8C), **ipallocd**(8C), **netconfig**(8C)

Finlayson, Ross, Timothy Mann, Jeffrey Mogul, and Marvin Theimer, *A Reverse Address Resolution Protocol*, RFC 903, Network Information Center, SRI International, Menlo Park, Calif., June 1984.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

rc, rc.boot, rc.local – command scripts for auto-reboot and daemons

SYNOPSIS

/etc/rc

/etc/rc.boot

/etc/rc.local

DESCRIPTION

rc and **rc.boot** are command scripts that are invoked by **init**(8) to perform file system housekeeping and to start system daemons. **rc.local** is a script for commands that are pertinent only to a specific site or client machine.

rc.boot sets the machine name, and then, if coming up multi-user, runs **fsck**(8) with the **–p** option. This "preens" the disks of minor inconsistencies resulting from the last system shutdown and checks for serious inconsistencies caused by hardware or software failure. If **fsck**(8) detects a serious disk problem, it returns an error and **init**(8) brings the system up in single-user mode. When coming up single-user, when **init**(8) is invoked by **fastboot**(8), or when it is passed the **–b** flag from **boot**(8S), functions performed in the **rc.local** file, including this disk check, are skipped.

Next, **rc** runs. If the system came up single-user, **rc** runs when the single-user shell terminates (see **init**(8)). It mounts 4.2 filesystems and spawns a shell for **/etc/rc.local**, which mounts NFS filesystems, and starts local daemons. After **rc.local** returns, **rc** starts standard daemons, preserves editor files, clears **/tmp**, starts system accounting (if applicable), starts the network (where applicable), and if enabled, runs **savecore**(8) to preserve the core image after a crash.

Sun386i

These files operate as described above with the following variations:

fsck(8) is invoked with the **–y** option to prevent users being put in single-user mode by happenstance.

rc.boot invokes **netconfig**(8C) to configure the system for the network before booting. **netconfig** is invoked before the **/usr** filesystem is mounted, because **/usr** might be mounted from a server. **netconfig** writes **/etc/net.conf** unless the **–n** option is specified, controlling system booting.

rc.boot dynamically loads device drivers.

rc invokes any programs found in **/var/recover** to clean up any operations partially completed when the system crashed or was shut down.

rc.local starts the automounter.

The file **/etc/net.conf** stores these environment variables: The **VERBOSE** environment variable controls the verbosity of the messages from the **rc** script; its value is taken from **NVRAM**. The **NETWORKED** environment variable controls whether services useful only on a networked system are started in **/etc/rc.local**. The **PNP** environment variable, set up during initial system installation, controls whether local network configuration information is used or whether that information comes from the network. (Using automatic system installation causes all systems except boot servers to get this information from the network, facilitating network reconfiguration.) The **HOSTNAME** and **DOMAINNAME** environment variables, used together, help determine if this system is a boot server or, with **PNP** set to **no**, control the host name and domain name.

FILES

/etc/rc
/etc/rc.boot
/etc/rc.local
/etc/net.conf
/var/recover/*
/var/yp/*
/tmp

SEE ALSO

automount(8), **boot**(8S), **fastboot**(8), **init**(8), **reboot**(8), **savecore**(8), **netconfig**(8C)

BUGS

The system message file **/var/adm/messages** is no longer created automatically.

NAME

rdate – set system date from a remote host

SYNOPSIS

/usr/ucb/rdate *hostname*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rdate sets the local date and time from the *hostname* given as argument. You must be super-user on the local system. Typically **rdate** can be inserted as part of your **/etc/rc.local** startup script.

FILES

/etc/rc.local

BUGS

Could be modified to accept a list of hostnames and try each until a valid date returned. Better yet would be to write a real date server that accepted broadcast requests.

NAME

reboot – restart the operating system

SYNOPSIS

/usr/etc/reboot [**−dnq**] [*boot arguments*]

DESCRIPTION

reboot executes the **reboot**(2) system call to restart the kernel. The kernel is loaded into memory by the PROM monitor, which transfers control to it. See **boot**(8S) for details.

Although **reboot** can be run by the super-user at any time, **shutdown**(8) is normally used first to warn all users logged in of the impending loss of service. See **shutdown**(8) for details.

reboot performs a **sync**(1) operation on the disks, and then a multiuser reboot is initiated. See **init**(8) for details.

reboot normally logs the reboot to the system log daemon, **syslogd**(8), and places a shutdown record in the login accounting file **/var/adm/wtmp**. These actions are inhibited if the **−n** or **−q** options are present.

Power Fail and Crash Recovery

Normally, the system will reboot itself at power-up or after crashes.

OPTIONS

−d Dump system core before rebooting.

−n Avoid the **sync**(1). It can be used if a disk or the processor is on fire.

−q Quick. Reboots quickly and ungracefully, without first shutting down running processes.

Boot Arguments

If a boot argument string is given, it is passed to the boot command in the PROM monitor. The string must be quoted if it contains spaces or other characters that could be interpreted by the shell. If the first character of the boot argument string is a minus sign '−' the string must be preceded by an option terminator string '−−' For example: 'reboot − − −s' to reboot and come up single user, '**reboot vmunix.test**' to reboot to a new kernel. See **boot**(8S) for details.

FILES

/var/adm/wtmp login accounting file

SEE ALSO

sync(1), **reboot**(2), **boot**(8S), **fsck**(8), **halt**(8), **init**(8), **panic**(8S), **shutdown**(8), **syslogd**(8)

NAME

renice – alter nice value of running processes

SYNOPSIS

/usr/etc/renice *priority pid*...

/usr/etc/renice *priority* [**–p** *pid*...] [**–g** *pgrp*...] [**–u** *username*...]

DESCRIPTION

renice alters the scheduling nice value, and hence the priority, of one or more running processes. See **nice**(1) for a discussion of nice value and process scheduling priority.

OPTIONS

By default, the processes to be affected are specified by their process IDs. *priority* is the new priority value.

–p *pid*... Specify a list of process IDs.

–g *pgrp*... Specify a list of process group IDs. The processes in the specified process groups have their scheduling priority altered.

–u *user*... Specify a list of user IDs or usernames. All processes owned by each *user* have their scheduling altered.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to 20. (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range –20 to 19. Useful nice values are 19 (the affected processes will run only when nothing else in the system wants to), 0 (the default nice value) and any negative value (to make things go faster).

If only the priority is specified, the current process (alternatively, process group or user) is used.

FILES

/etc/passwd to map user names to user ID's

SEE ALSO

pstat(8)

BUGS

If you make the nice value very negative, then the process cannot be interrupted.

To regain control you must make the priority greater than zero.

Users other than the super-user cannot increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

repquota – summarize quotas for a file system

SYNOPSIS

/usr/etc/repquota [**–v**] *filesystem*...

/usr/etc/repquota [**–av**]

DESCRIPTION

repquota prints a summary of the disc usage and quotas for the specified file systems. For each user the current number of files and amount of space (in kilobytes) is printed, along with any quotas created with **edquota**(8).

OPTIONS

–a Report on all file systems indicated in **/etc/fstab** to be read-write with quotas.

–v Report all quotas, even if there is no usage.

Only the super-user may view quotas which are not their own.

FILES

quotas	quota file at the file system root
/etc/fstab	default file systems

SEE ALSO

quota(1), **quotactl**(2), **edquota**(8), **quotacheck**(8), **quotaon**(8)

NAME

restore, rrestore – incremental file system restore

SYNOPSIS

/usr/etc/restore –irRtx [*filename* ...]

DESCRIPTION

restore restores files from backup tapes created with the **dump**(8) command. *options* is a string of at least one of the options listed below, along with any modifiers and arguments you supply. Remaining arguments to **restore** are the names of files (or directories whose files) are to be restored to disk. Unless the **h** modifier is in effect, a directory name refers to the files it contains, and (recursively) its subdirectories and the files they contain.

OPTIONS

i Interactive. After reading in the directory information from the tape, **restore** invokes an interactive interface that allows you to browse through the dump tape's directory hierarchy, and select individual files to be extracted. See **Interactive Commands**, below, for a description of available commands.

r Restore the entire tape. Load the tape's full contents into the current directory. This option should only be used to restore a complete dump tape onto a clear filesystem, or to restore an incremental dump tape after a full "level **0**" restore. For example:

```
example# /usr/etc/newfs /dev/rxy0g
example# /usr/etc/mount /dev/xy0g /mnt
example# cd /mnt
example# restore r
```

is a typical sequence to restore a "level **0**" dump. Another **restore** can be done to get an incremental dump in on top of this.

R Resume restoring. **restore** requests a particular tape of a multivolume set from which to resume a full restore (see the **r** option above). This allows **restore** to start from a checkpoint when it is interrupted in the middle of a full restore.

t Table of contents. List each *filename* that appears on the tape. If no *filename* argument is given, the root directory is listed. This results in a list of all files on the tape, unless the **h** modifier is in effect. (The **t** option replaces the function of the old **dumpdir** program).

x Extract the named files from the tape. If a named file matches a directory whose contents were written onto the tape, and the **h** modifier is not in effect, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no *filename* argument is given, the root directory is extracted. This results in the entire tape being extracted unless the **h** modifier is in effect.

Modifiers

Some of the following modifiers take arguments that are given as separate words on the command line. When more than one such modifier appears within *options*, the arguments must appear in the same order as the modifiers that they apply to.

a *archive-file*
The dump table of contents is taken from the specified *archive-file* instead of from a dump tape. If a requested file is present in the table of contents, *restore* will prompt for the tape volume to be mounted. If only contents information is needed, for example when the *t* option is specified, or the *i* option is specified without a corresponding *extract* request, no dump tape will have to be mounted.

c Convert the contents of the dump tape to the new filesystem format.

d Debug. Turn on debugging output.

h Extract the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

m Extract by inode numbers rather than by filename to avoid regenerating complete pathnames. This is useful if only a few files are being extracted.

v Verbose. **restore** displays the name of each file it restores, preceded by its file type.

y Do not ask whether to abort the restore in the event of tape errors. **restore** tries to skip over the bad tape block(s) and continue as best it can.

b *factor*
Blocking factor. Specify the blocking factor for tape reads. By default, **restore** will attempt to figure out the block size of the tape. Note: a tape block is 512 bytes.

f *dump-file*
Use *dump-file* instead of **/dev/rmt?** as the file to restore from. If *dump-file* is specified as '–', **restore** reads from the standard input. This allows, **dump**(8) and **restore** to be used in a pipeline to dump and restore a file system:

example# dump 0f – /dev/rxy0g | (cd /mnt; restore xf –)

If the name of the file is of the form *machine*:*device* the restore is done from the specified machine over the network using **rmt**(8C). Since **restore** is normally run by root, the name of the local machine must appear in the **.rhosts** file of the remote machine. If the file is specified as *user@machine*:*device*, **restore** will attempt to execute as the specified user on the remote machine. The specified user must have a **.rhosts** file on the remote machine that allows root from the local machine.

s *n* Skip to the *n*'th file when there are multiple dump files on the same tape. For example, the command:

example# restore xfs /dev/nrar0 5

would position you at the fifth file on the tape.

USAGE

Interactive Commands

restore enters interactive mode when invoked with the **i** option. Interactive commands are reminiscent of the shell. For those commands that accept an argument, the default is the current directory.

ls [*directory*]
List files in *directory* or the current directory, represented by a '.' (period). Directories are appended with a '/' (slash). Entries marked for extraction are prefixed with a '*' (asterisk). If the verbose option is in effect, inode numbers are also listed.

cd *directory*
Change to directory *directory* (within the dump-tape).

pwd Print the full pathname of the current working directory.

add [*filename*]
Add the current directory, or the named file or directory **directory** to the list of files to extract. If a directory is specified, add that directory and its files (recursively) to the extraction list (unless the **h** modifier is in effect).

delete [*filename*]
Delete the current directory, or the named file or directory from the list of files to extract. If a directory is specified, delete that directory and all its descendents from the extraction list (unless the **h** modifier is in effect). The most expedient way to extract a majority of files from a directory is to add that directory to the extraction list, and then delete specific files to omit.

extract Extract all files on the extraction list from the dump tape. **restore** asks which volume the user wishes to mount. The fastest way to extract a small number of files is to start with the last tape volume and work toward the first.

verbose Toggle the status of the **v** modifier. While **v** is in effect, the **ls** command lists the inode numbers of all entries, and **restore** displays information about each file as it is extracted.

help Display a summary of the available commands.

quit **restore** exits immediately, even if the extraction list is not empty.

FILES

/dev/rmt8	the default tape drive
dumphost:/dev/rmt8	the default tape drive if called as **rrestore**
/tmp/rstdir*	file containing directories on the tape
/tmp/rstmode*	owner, mode, and timestamps for directories
./restoresymtable	information passed between incremental restores

SEE ALSO

dump(8), **mkfs**(8), **mount**(8), **newfs**(8), **rmt**(8C)

DIAGNOSTICS

restore complains about bad option characters.

Read errors result in complaints. If **y** has been specified, or the user responds **y**, **restore** will attempt to continue.

If the dump extends over more than one tape, **restore** asks the user to change tapes. If the **x** or **i** option has been specified, **restore** also asks which volume the user wishes to mount.

There are numerous consistency checks that can be listed by **restore**. Most checks are self-explanatory or can "never happen". Common errors are given below.

Converting to new file system format.
A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

filename: **not found on tape**
The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file *inumber*, **got** *inumber*
A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
When doing an incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or one that has too high an incremental level has been loaded.

Tape read error while restoring *filename*
Tape read error while skipping over inode inumber
Tape read error while trying to resynchronize
A tape read error has occurred.
If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped *num* blocks

After a tape read error, **restore** may have to resynchronize itself. This message lists the number of blocks that were skipped over.

BUGS

restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A "level **0**" dump must be done after a full restore. Because **restore** runs in user mode, it has no control over inode allocation; this means that **restore** repositions the files, although it does not change their contents. Thus, a full dump must be done to get a new set of directories reflecting the new file positions, so that later incremental dumps will be correct.

NAME

rexd, rpc.rexd – RPC-based remote execution server

SYNOPSIS

/usr/etc/rpc.rexd [–s]

DESCRIPTION

rexd is the Sun RPC server for remote program execution. This daemon is started by **inetd**(8C) whenever a remote execution request is made.

For noninteractive programs, the standard file descriptors are connected directly to TCP connections. Interactive programs involve pseudo-terminals, in a fashion that is similar to the login sessions provided by **rlogin**(1C). This daemon may use NFS to mount file systems specified in the remote execution request.

FILES

/dev/ttyp*n*	pseudo-terminals used for interactive mode
/etc/passwd	authorized users
/tmp_rex/rexd??????	temporary mount points for remote file systems.

OPTIONS

–s Secure. When specified, requests must have valid des credentials. If the request does not have a DES credential it is rejected. The default publickey credential is rejected. Only newer **on** commands send DES credentials.

If access is denied with an Authentication error, you may have to set your publickey with the **chkey**(1) command.

SEE ALSO

chkey(1), **on**(1C), **rlogin**(1C), **rex**(3R), **exports**(5), **inetd.conf**(5), **publickey**(5), **inetd**(8C)

DIAGNOSTICS

Diagnostic messages are normally printed on the console, and returned to the requestor.

RESTRICTIONS

Root cannot execute commands using **rexd** client programs such as **on**(1C).

NAME

rexecd, in.rexecd – remote execution server

SYNOPSIS

/usr/etc/in.rexecd *host.port*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rexecd is the server for the **rexec**(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by **inetd**(8C), and then executes the following protocol:

- The server reads characters from the socket up to a null (**\0**) byte. The resultant string is interpreted as an ASCII number, base 10.
- If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.
- A null terminated user name of at most 16 characters is retrieved on the initial socket.
- A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.
- A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
- **rexecd** then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.
- A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rexecd**.

SEE ALSO

rexec(3N) **inetd**(8C)

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

username too long
The name is longer than 16 characters.

password too long
The password is longer than 16 characters.

command too long
The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect.
No password file entry for the user name existed.

Password incorrect.
The wrong password was supplied.

No remote directory.
The **chdir** command to the home directory failed.

Try again.
A **fork** by the server failed.

/usr/bin/sh: . . .
The user's login shell could not be started.

BUGS

Indicating '**Login incorrect**' as opposed to '**Password incorrect**' is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rfadmin – RFS domain administration

SYNOPSIS

rfadmin
rfadmin –p
rfadmin –a *hostname*
rfadmin –r *hostname*

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rfadmin is used to add and remove hosts and their associated authentication information from a *domain*/**passwd** file on a Remote File Sharing (RFS) primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, **rfadmin** returns the *hostname* of the current domain name server for the local domain. For each *domain*, **/usr/nserve/auth.info/***domain***/passwd** is created on the primary, and should be copied to all secondaries, and all hosts that want to do password verification of hosts in the *domain*.

rfadmin can only be used to modify domain files on the primary domain name server (**–a** and **–r** options). If domain name server reponsibilities are temporarily passed to a secondary domain name server, that computer can use the **–p** option to pass domain name server responsibility back to the primary. Any host can use **rfadmin** with no options to print information about the domain. The user must have **root** permissions to use the command.

Using **rfadmin** with the **–a** option, will result in an error if *hostname* is not unique in the domain.

Using **rfadmin** with the **–r** option, will send an error to the standard error if one of the following is true:

- *hostname* does not exist in the domain.
- *hostname* is defined as a domain name server.
- There are resources advertised by *hostname*.

When used with the **–p** option, **rfadmin** sends an error message to standard error, if there are no backup name servers defined for *domain*.

OPTIONS

–p Pass the domain name server responsibilities back to a primary or to a secondary name server.

–a *hostname*
Add a host to a domain that is served by this domain name server. *hostname* must be of the form *domain.nodename*. Create an entry for *hostname* in the *domain*/**passwd** file, which has the same format as **/etc/passwd**, and prompt for an initial authentication password; the password prompting process conforms with that of **passwd**(1).

–r *hostname*
Remove a host from its domain by removing it from the *domain*/**passwd** file.

FILES

/usr/nserve/auth.info/*domain***/passwd**

SEE ALSO

passwd(1), **mount**(8), **rfstart**(8), **rfstop**(8)

NAME

rfpasswd – change RFS host password

SYNOPSIS

rfpasswd

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rfpasswd updates the Remote File Sharing (RFS) authentication password for a host; processing of the new password follows the same criteria as **passwd**(1). The updated password is registered at the domain name server (**/usr/nserve/auth.info/***domain***/passwd**) and replaces the password stored at the local host (**/usr/nserve/loc.passwd/file**).

This command is restricted to the super-user.

Note: if you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain/passwd* file from the domain's primary name server. If this is not done, *attempts to mount remote resources may fail*.

If any of the following is true an error message will be sent to the standard error:

- The old password entered from this command does not match the existing password for this machine.
- The two new passwords entered from this command do not match.
- The new password does not satisfy the security criteria in **passwd**(1).
- The domain name server does not know about this machine.
- The command is not run with super-user privileges.

Also, RFS must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

/usr/nserve/auth.info/*domain***/passwd**
/usr/nserve/loc.passwd

SEE ALSO

passwd(1), **rfadmin**(8), **rfstart**(8)

NAME

rfstart – start RFS

SYNOPSIS

rfstart [**–v**] [**–p** *primary_addr*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rfstart starts Remote File Sharing (RFS) and defines an authentication level for incoming requests. This command can be used only after the domain name server is set up and your computer's domain name and network specification has been defined using **dname**(8).

If the host password has not been set, **rfstart** will prompt for a password; the password prompting process must match the password entered for your machine at the primary domain name server (see **rfadmin**(8)). If you remove the **loc.passwd** file or change domains, you will also have to reenter the password.

Also, when **rfstart** is run on a domain name server, entries in the **rfmaster**(5) file are syntactically validated.

This command is restricted to the super-user.

If syntax errors are found in validating the **rfmaster**(5) file, a warning describing each error will be sent to the standard error.

An error message will be sent to the standard error if any of the following is true:

- The shared resource environment is already running.
- There is no communications network.
- The domain name server cannot be found.
- The domain name server does not recognize the machine.
- The command is run without super-user privileges.

Remote file sharing will not start if the host password in **/usr/nserve/loc.passwd** is corrupted. If you suspect this has happened, remove the file and run **rfstart** again to reenter your password.

Note: **rfstart** will *not* fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary or any other host that validates your password, the mount will fail if your password does not match the one that host has listed for your machine.

OPTIONS

–v Specify that verification of all clients is required in response to initial incoming mount requests; any host not in the file **/usr/nserve/auth.info/***domain***/passwd** for the **domain** they belong to, will not be allowed to mount resources from your host. If the –**v** option is not specified, hosts named in *domain***/passwd** will be verified, other hosts will be allowed to connect without verification.

–p *primary_addr*
Indicate the primary domain name server for your domain. *primary_addr* must be the network address of the primary name server for your domain. If the **–p** option is not specified, the address of the domain name server is taken from the **rfmaster** file. See **rfmaster**(5) for a description of the valid address syntax.

FILES

/usr/nserve/rfmaster
/usr/nserve/loc.passwd

SEE ALSO

rfmaster(5), **adv**(8), **dname**(8), **mount**(8), **rfadmin**(8), **rfstop**(8), **unadv**(8)

NAME

rfstop – stop the RFS environment

SYNOPSIS

rfstop

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rfstop disconnects a host from the Remote File Sharing (RFS) environment until another **rfstart**(8) is executed.

When executed on the domain name server, the domain name server responsibility is moved to a secondary name server as designated in the **rfmaster** file.

This command is restricted to the super-user.

If any of the following is true, an error message will be sent to standard error.

- There are resources currently advertised by this host.
- Resources from this machine are still remotely mounted by other hosts.
- There are still remotely mounted resources in the local file system tree.
- **rfstart**(8) had not previously been executed.
- The command is not run with super-user privileges.

SEE ALSO

rfmaster(5), **adv**(8), **mount**(8), **rfadmin**(8), **rfstart**(8), **unadv**(8)

NAME

rfuadmin – RFS notification shell script

SYNOPSIS

rfuadmin *message remote_resource* [*seconds*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

The **rfuadmin** shell script is used to respond to unexpected Remote File Sharing (RFS) events picked up by the **rfudaemon**(8) process. Such events may include broken network connections and forced unmounts. This script is not intended to be run directly from the shell.

Responses to messages received by **rfudaemon** can be tailored to suit the particular system by editing this script. The following paragraphs describe the arguments passed to **rfuadmin** and its standard responses.

disconnect *remote_resource*

A link to a remote resource has been cut. **rfudaemon** executes **rfuadmin,** passing it the message **disconnect** and the name of the disconnected resource. **rfuadmin** sends this message to all terminals using **wall**(1):

remote_resource **has been disconnected from the system.**

rfuadmin executes **fuser**(8) to kill all processes using the resource, unmounts the resource, and attempts to mount the resource again.

fumount remote_resource

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

fuwarn *remote_resource seconds*

This message notifies **rfuadmin** that a resource is about to be unmounted. **rfudaemon** sends this script the **fuwarn** message, the resource name, and the number of seconds in which the forced unmount will occur. **rfuadmin** sends this message to all terminals:

remote_resource **is being removed from the system in # seconds.**

SEE ALSO

wall(1), **fumount**(8), **fuser**(8), **mount**(8), **rfstart**(8), **rfudaemon**(8)

BUGS

The console must be on when RFS is running, otherwise **rfuadmin** hangs when it attempts to write to it, in which case recovery from disconected resources may not complete.

NAME

rfudaemon – Remote File Sharing daemon

SYNOPSIS

rfudaemon

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

The RFS daemon, **rfudaemon**, is started automatically by **rfstart**(8) and runs as a daemon process while Remote File Sharing is active. It listens for unexpected events, such as broken network connections and forced unmounts, and invokes **rfuadmin**(8) to execute the appropriate administrative procedures. Events recognized by **rfudaemon** are as follows:

disconnect
A link to a remote resource has been cut. **rfudaemon** executes **rfuadmin**, with two arguments: **disconnect** and the name of the disconnected resource.

fumount
A remote server machine has forced an unmount of a resource a local machine has mounted. **rfudaemon** executes **rfuadmin**, with two arguments: **fumount** and the name of the disconnected resource.

getumsg
A remote user-level program has sent a message to the local **rfudaemon**. Currently the only message sent is **fuwarn**, which notifies **rfuadmin** that a resource is about to be unmounted. **rfudaemon** sends **rfuadmin** the **fuwarn**, the resource name, and the number of seconds in which the forced unmount will occur.

lastumsg
The local machine wants to stop the **rfudaemon** (**rfstop**(8)). This causes **rfudaemon** to exit.

SEE ALSO

rfstart(8), **rfstop**(8), **rfuadmin**(8)

NAME

rlogind, in.rlogind – remote login server

SYNOPSIS

/usr/etc/in.rlogind *host.port*

DESCRIPTION

rlogind is the server for the **rlogin**(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

rlogind is invoked by **inetd**(8C) when a remote login connection is established, and executes the following protocol:

- The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to **rlogind** by **inetd** in the form *host.port* with host in hex and port in decimal.
- The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see **hosts**(5)), the server aborts the connection.

Once the source port and address have been checked, **rlogind** allocates a pseudo-terminal (see **pty**(4)), and manipulates file descriptors so that the slave half of the pseudo-terminal becomes the **stdin**, **stdout**, and **stderr** for a login process. The login process is an instance of the **login**(1) program, invoked with the **–r** option. The login process then proceeds with the authentication process as described in **rshd**(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo-terminal, operating as an intermediary between the login process and the client instance of the **rlogin** program. In normal operation, the packet protocol described in **pty**(4) is invoked to provide **^S/^Q** type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, **TERM**; see **environ**(5V).

SEE ALSO

inetd(8C)

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

Hostname for your address unknown.
No entry in the host name database existed for the client's machine.

Try again.
A *fork* by the server failed.

/usr/bin/sh: . . .
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rmail – handle remote mail received via uucp

SYNOPSIS

rmail *recipient*. . .

DESCRIPTION

rmail interprets incoming mail received through **uucp**(1C), collapsing ''From'' lines in the form generated by **bin-mail (1)** (see **bin-mail**(1)) into a single line of the form *return-path*!*sender*, and passing the processed mail on to **sendmail**(8).

rmail is explicitly designed for use with **uucp**(1C) and **sendmail**(8).

SEE ALSO

bin-mail(1), **uucp**(1C), **sendmail**(8)

NAME

rm_client – remove an NFS client

SYNOPSIS

rm_client [**–y**] *clients*

DESCRIPTION

rm_client removes an NFS client from a server. By default, **rm_client** asks if you want to remove the client's root directory, swap file, hosts entry, and **/tftpboot** file and whether to delete the client's entry in **/etc/bootparams**. **rm_client** can be run only by the super-user on the server, while in multiuser mode, or while not in the miniroot.

OPTIONS

–y Supply "yes" answers to all questions about what to remove.

FILES

/etc/bootparams
/tftpboot/*machine_addr*
/export/root/*client*
/export/swap/*client*

SEE ALSO

add_client(8), **add_services**(8), **suninstall**(8)

Installing SunOS 4.1

DIAGNOSTICS

must be run as root (super-user).
You must be root to run **rm_client**.

NAME

rmntstat – display RFS mounted resource information

SYNOPSIS

rmntstat [**–h**] [*resource*]

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

When used with no options, **rmntstat** displays a list of all local Remote File Sharing resources that are remotely mounted, the local path name, and the corresponding clients. **rmntstat** returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, **rmntstat** displays the remote mount information only for that resource.

This command is restricted to the super-user.

OPTIONS

–h Omit header information from the display.

EXIT STATUS

If no local resources are remotely mounted, **rmntstat** will return a successful exit status.

ERRORS

If *resource* does not physically reside on the local machine or is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

mount(8), **fumount**(8), **unadv**(8)

NAME

rmt – remote magtape protocol module

SYNOPSIS

/usr/etc/rmt

DESCRIPTION

rmt is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. **rmt** is normally started up with an **rexec**(3N) or **rcmd**(3N) call.

The **rmt** program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

> **A***number\n*

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

> **E***error-number***\n***error-message***\n**

where *error-number* is one of the possible error numbers described in **intro**(2) and *error-message* is the corresponding error string as printed from a call to **perror**(3). The protocol is comprised of the following commands:

S — Return the status of the open device, as obtained with a **MTIOCGET ioctl** call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

C*device* — Close the currently open device. The *device* specified is ignored.

I*operation***\n***count***\n** — Perform a **MTIOCOP ioctl**(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the **ioctl** call. The return value is the *count* parameter when the operation is successful.

L*whence***\n***offset***\n** — Perform an **lseek**(2V) operation using the specified parameters. The response value is that returned from the **lseek** call.

O*device***\n***mode***\n** — Open the specified *device* using the indicated *mode*. *device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to **open**(2V). If a device had already been opened, it is closed before a new open is performed.

R*count* — Read *count* bytes of data from the open device. **rmt** performs the requested **read**(2V) and responds with **A***count-read\n* if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

W*count* — Write data onto the open device. **rmt** reads *count* bytes from the connection, aborting if a premature EOF is encountered. The response value is that returned from the **write**(2V) call.

Any other command causes **rmt** to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

intro(2), **ioctl**(2), **lseek**(2V), **open**(2V), **read**(2V), **write**(2V), **perror**(3), **rcmd**(3N), **rexec**(3N), **mtio**(4), **dump**(8), **restore**(8)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/usr/etc/route [**–fn**] **add** | **delete** [**host** | **net**] *destination* [*gateway* [*metric*]]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

route manually manipulates the network routing tables normally maintained by the system routing daemon, **routed**(8C), or through default routes and redirect messages from routers. **route** allows the super-user to operate directly on the routing table for the specific host or network indicated by *destination*. The *gateway* argument, if present, indicates the network gateway to which packets should be addressed. The *metric* argument indicates the number of "hops" to the *destination*. The *metric* is required for *add* commands; it must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways.

The **add** command instructs **route** to add a route to *destination*. **delete** deletes a route.

Routes to a particular host must be distinguished from those to a network. The optional keywords **net** and **host** force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a "local address part" of **INADDR_ANY**, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected by a gateway, the *metric* parameter should be greater than 0. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used directly for transmission. All symbolic names specified for a *destination* or *gateway* are looked up in the hosts database using **gethostbyname( )** (see **gethostent**(3N)). If this lookup fails, then the name is looked up in the networks database using **getnetbyname( )** (see **getnetent**(3N)). "default" is also a valid destination, which is used for all routes if there is no specific host or network route.

OPTIONS

–f Flush the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, **route** flushes the gateways before performing the command.

–n Prevents attempts to print host and network names symbolically when reporting actions. This is useful, for example, when all name servers are down on your local net, so you need a route before you can contact the name server.

FILES

/etc/hosts
/etc/networks

SEE ALSO

ioctl(2), **gethostent**(3N), **getnetent**(3N), **routing**(4N), **routed**(8C)

DIAGNOSTICS

add [**host** | **net**] *destination*:*gateway*

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the **ioctl**(2) call.

delete [**host** | **net**] *destination*:*gateway*

The specified route is being deleted.

*destination***done**

When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

Network is unreachable

An attempt to add a route failed because the gateway listed was not on a directly-connected network. Give the next-hop gateway instead.

not in table

A delete operation was attempted for an entry that is not in the table.

routing table overflow

An add operation was attempted, but the system was unable to allocate memory to create the new entry.

NAME

routed, in.routed – network routing daemon

SYNOPSIS

/usr/etc/in.routed [**–qstv**] [*logfile*]

DESCRIPTION

routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation **routed** listens on **udp**(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When **routed** is started, it uses the **SIOCGIFCONF ioctl( )** (see **ioctl**(2)) to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. **routed** then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, **routed** formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

request packets received by **routed** are used to update the routing tables if one of the following conditions is satisfied:

- No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (that is, the hop count is not infinite).
- The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.
- The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.
- The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, **routed** records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. **routed** waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, **routed** also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

In addition to the facilities described above, **routed** supports the notion of "distant" *passive* and *active* gateways. When **routed** is started up, it reads the file **/etc/gateways** to find gateways which may not be identified using the **SIOGIFCONF ioctl( )**. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a **routed** process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The **/etc/gateways** is comprised of a series of lines, each in the following format:

> **< net | host >** *filename1* **gateway** *filename2* **metric** *value* **< passive | active >**

The **net** or **host** keyword indicates if the route is to a network or specific host.

filename1 is the name of the destination network or host. This may be a symbolic name located in **/etc/networks** or **/etc/hosts**, or an Internet address specified in "dot" notation; see **inet**(3N).

filename2 is the name or address of the gateway to which messages should be forwarded.

value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as passive or active (as described above).

OPTIONS

–s Force **routed** to supply routing information whether it is acting as an internetwork router or not.

–q Opposite of the **–s** option.

–t All packets sent or received are printed on the standard output. In addition, **routed** will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process.

–v Allow a logfile to be created showing the changes made to the routing tables with a timestamp.

logfile Specify a file in which **routed** records any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

FILES

/etc/gateways	for distant gateways
/etc/networks	
/etc/hosts	

SEE ALSO

ioctl(2), **inet**(3N), **udp**(4P)

BUGS

The kernel's routing tables may not correspond to those of **routed** for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

rpcinfo – report RPC information

SYNOPSIS

rpcinfo –p [*host*]

rpcinfo [**–n** *portnum*] **–u** *host program* [*version*]

rpcinfo [**–n** *portnum*] **–t** *host program* [*version*]

rpcinfo –b *program version*

rpcinfo –d *program version*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rpcinfo makes an RPC call to an RPC server and reports what it finds.

OPTIONS

–p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by **hostname**(1).

–u Make an RPC call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

–t Make an RPC call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.

–n Use *portnum* as the port number for the *–t* and *–u* options instead of the port number given by the portmapper.

–b Make an RPC broadcast to procedure 0 of the specified *program* and *version* using UDP and report all hosts that respond.

–d Delete registration for the RPC service of the specified *program* and *version*. This option can be exercised only by the super-user.

The *program* argument can be either a name or a number.

If a *version* is specified, **rpcinfo** attempts to call that version of the specified *program*. Otherwise, **rpcinfo** attempts to find all the registered version numbers for the specified *program* by calling version 0 (which is presumed not to exist; if it does exist, **rpcinfo** attempts to obtain this information by calling an extremely high version number instead) and attempts to call each registered version. Note: the version number is required for **–b** and **–d** options.

EXAMPLES

To show all of the RPC services registered on the local machine use:

example% rpcinfo –p

To show all of the RPC services registered on the machine named **klaxon** use:

example% rpcinfo –p klaxon

To show all machines on the local net that are running the Network Interface Service (NIS) use:

example% rpcinfo –b ypserv 'version' | uniq

where 'version' is the current NIS version obtained from the results of the **–p** switch above.

To delete the registration for version 1 of the **walld** service use:

example% rpcinfo –d walld 1

SEE ALSO

rpc(5), **portmap**(8C)

RPC Programming Guide in *Network Programming*

BUGS

In releases prior to the SunOS 3.0 release, the Network File System (NFS) did not register itself with the portmapper; **rpcinfo** cannot be used to make RPC calls to the NFS server on hosts running such releases.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

rquotad, rpc.rquotad – remote quota server

SYNOPSIS

/usr/etc/rpc.rquotad

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rquotad is an **rpc**(3N) server which returns quotas for a user of a local file system which is mounted by a remote machine over the NFS. The results are used by **quota**(1) to display user quotas for remote file systems. The **rquotad** daemon is normally invoked by **inetd**(8C).

FILES

quotas	quota file at the file system root

SEE ALSO

quota(1), **rpc**(3N), **nfs**(4P), **services**(5) **inetd**(8C)

NAME

rshd, in.rshd – remote shell server

SYNOPSIS

/usr/etc/in.rshd *host.port*

DESCRIPTION

rshd is the server for the **rcmd**(3N) routine and, consequently, for the **rsh**(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

rshd is invoked by **inetd**(8C) each time a shell service is requested, and executes the following protocol:

- The server checks the client's source port. If the port is not in the range 512-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to **rshd.**
- The server reads characters from the socket up to a null (**0**) byte. The resultant string is interpreted as an ASCII number, base 10.
- If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 512-1023.
- The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see **hosts**(5)), the server aborts the connection.
- A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.
- A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.
- A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
- **rshd** then validates the user according to the following steps. The remote user name is looked up in the password file and a **chdir** is performed to the user's home directory. If the lookup or fails, the connection is terminated. If the **chdir** fails, it does a **chdir** to / **(root).** If the user is not the super-user, (user ID 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file **.rhosts** in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.
- A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rshd.**

FILES

/etc/hosts.equiv

SEE ALSO

rsh(1C), **rcmd**(3N), **syslogd**(8)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

DIAGNOSTICS

The following diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

locuser too long
The name of the user on the client's machine is longer than 16 characters.

remuser too long
The name of the user on the remote machine is longer than 16 characters.

command too long
The command line passed exceeds the size of the argument list (as configured into the system).

Hostname for your address unknown.
No entry in the host name database existed for the client's machine.

Login incorrect.
No password file entry for the user name existed.

Permission denied.
The authentication procedure described above failed.

Can't make pipe.
The pipe needed for the **stderr**, was not created.

Try again.
A *fork* by the server failed.

/usr/bin/sh: ...
The user's login shell could not be started.

In addition, daemon's status messages and internal diagnostics are logged to the appropriate system log using the **syslogd**(8) facility.

NAME

rstatd, rpc.rstatd – kernel statistics server

SYNOPSIS

/usr/etc/rpc.rstatd

DESCRIPTION

rstatd is a server which returns performance statistics obtained from the kernel. These statistics are graphically displayed by **perfmeter**(1). The **rstatd** daemon is normally invoked by **inetd**(8C).

Systems with disk drivers to be monitored by this daemon must be configured so as to report disk (**_dk_xfer**) statistics.

SEE ALSO

perfmeter(1), **services**(5), **inetd**(8C)

NAME

runacct – run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [*mmdd* [*state*]]

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated using **cron**(8). **runacct** processes connect, fee, disk, and process accounting files. It also prepares summary files for **prdaily** or billing purposes.

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see **mail**(1)) is sent to **root**, and **runacct** terminates. **runacct** uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. **runacct** then looks in **statefile** to see what it has done and to determine what to process next. *states* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of the **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of **chargefee** into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/var/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart **runacct** after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files, such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before **runacct** can be restarted. The argument *mmdd* is necessary if **runacct** is being restarted, and specifies the month and day for which **runacct** will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start **runacct**:

nohup runacct 2> /var/adm/acct/nite/fd2log &

To restart **runacct**:

nohup runacct 0601 2>> /var/adm/acct/nite/fd2log &

To restart **runacct** at a specific *state*:

nohup runacct 0601 MERGE 2>> /var/adm/acct/nite/fd2log &

FILES

/etc/wtmp
/var/adm/pacct*
/var/adm/acct/nite/active
/var/adm/acct/nite/daytacct
/var/adm/acct/nite/lock
/var/adm/acct/nite/lock1
/var/adm/acct/nite/lastdate
/var/adm/acct/nite/statefile
/var/adm/acct/nite/ptacct*.*_mmdd_

SEE ALSO

acctcom(1), **mail**(1), **acct**(2V), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **cron**(8), **fwtmp**(8)

BUGS

Normally it is not a good idea to restart **runacct** in the **SETUP** *state*. Run **SETUP** manually and restart using:

runacct *mmdd* **WTMPFIX**

If **runacct** failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

rusage – print resource usage for a command

SYNOPSIS

rusage *command*

DESCRIPTION

The **rusage** command is similar to **time**(1V). It runs the given *command*, which must be specified; that is, *command* is not optional as it is in the C shell's timing facility. When the command is complete, **rusage** displays the real (wall clock), the system CPU, and the user CPU times which elapsed during execution of the command, plus other fields in the **rusage** structure, all on one long line. Times are reported in seconds and hundredths of a second.

EXAMPLE

The example below shows the format of **rusage** output.

```
example% rusage wc /usr/man/man1/csh (1)
3045   13423   78071 /usr/man/man1/csh (1)
2.26 real 0.80 user 0.36 sys 11 pf 38 pr 0 sw 11 rb 0 wb 16 vcx 37 icx 24 mx 0 ix 1230 id 9 is
example%
```

Each of the fields identified corresponds to an element of the **rusage** structure, as described in **getrusage**(2), as follows:

real		elapsed real time
user	ru_utime	user time used
sys	ru_stime	system time used
pf	ru_majflt	page faults requiring physical I/O
pr	ru_minflt	page faults not requiring physical I/O
sw	ru_nswap	swaps
rb	ru_inblock	block input operations
wb	ru_oublock	block output operations
vcx	ru_nvcsw	voluntary context switches
icx	ru_nivcsw	involuntary context switches
mx	ru_maxrss	maximum resident set size
ix	ru_ixrss	currently 0
id	ru_idrss	integral resident set size
is	ru_isrss	currently 0

SEE ALSO

csh(1), **time**(1V), **getrusage**(2)

BUGS

When the command being timed is interrupted, the timing values displayed may be inaccurate.

NAME

rusersd, rpc.rusersd – network username server

SYNOPSIS

/usr/etc/rpc.rusersd

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rusersd is a server that returns a list of users on the network. The **rusersd** daemon is normally invoked by **inetd**(8C).

SEE ALSO

perfmeter(1), **rusers**(1C), **services**(5) **inetd**(8C)

Installing SunOS 4.1

NAME

rwalld, rpc.rwalld – network rwall server

SYNOPSIS

/usr/etc/rpc.rwalld

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rwalld is a server that handles **rwall**(1C) and **shutdown**(2) requests. It is implemented by calling **wall**(1) to all the appropriate network machines. The **rwalld** daemon is normally invoked by **inetd**(8C).

SEE ALSO

rwall(1C), **wall**(1), **shutdown**(2) **services**(5), **inetd**(8C)

NAME

rwhod, in.rwhod – system status server

SYNOPSIS

/usr/etc/in.rwhod

AVAILABILITY

Due to its potential impact on network performance, this service is commented out of the **/etc/rc** system initialization script. It is provided only for 4.3 BSD compatibility.

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rwhod is the server which maintains the database used by the **rwho**(1C) and **ruptime**(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other **rwhod** servers' status messages, validating them, then recording them in a collection of files located in the directory **/var/spool/rwho**.

The **rwho** server transmits and receives messages at the port indicated in the "rwho" service specification, see **services**(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];     /* tty name */
        char    out_name[8];     /* user id */
        long    out_time;        /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
                struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the **w**(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the **gethostname**(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the **utmp**(5V) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the **rwho** server are discarded unless they originated at a **rwho** server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by **rwhod** are placed in files named **whod.***hostname* in the directory **/var/spool/rwho**. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. **rwhod** performs an **nlist (3V)** on **/vmunix** every 10 minutes to guard against the possibility that this file is not the system image currently operating.

FILES

/etc/rc
/var/spool/rwho

SEE ALSO

rwho(1C), **ruptime**(1C), **w**(1), **gethostname**(2), **nlist**(3V), **utmp**(5V), **syslogd**(8)

DIAGNOSTICS

Status and diagnostic messages are logged to the appropriate system log using the **syslogd**(8) facility.

BUGS

This service takes up progressively more network bandwidth as the number of hosts on the local net increases. For large networks, the cost becomes prohibitive. RPC-based services such as **rup**(1C) and **rusers**(1C) provide a similar function with greater efficiency.

rwhod should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

sa, accton – system accounting

SYNOPSIS

/usr/etc/sa [**–abcdDfijkKlmnrstu**] [–v[*n*]] [**–S** *savacctfile*] [**–U** *usracctfile*] [*filename*]

/usr/lib/acct/accton [*filename*]

DESCRIPTION

With an argument naming an existing *filename*, **accton** causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

sa reports on, cleans up, and generally maintains accounting files.

sa is able to condense the information in **/var/adm/pacct** into a summary file **/var/adm/savacct** which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system **/var/adm/pacct** can grow by 500K bytes per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; **/var/adm/pacct** is the default.

Output fields are labeled: **cpu** for the sum of user+system time (in minutes), **re** for real time (also in minutes), **k** for CPU-time averaged core usage (in 1k units), **avio** for average number of I/O operations per execution. With options fields labeled **tio** for total I/O operations, **k*sec** for CPU storage integral (kilo-core seconds), **u** and s for user and system CPU time alone (both in minutes) will sometimes appear.

sa also breaks out accounting statistics by user. This information is kept in the file **/var/adm/usracct**.

OPTIONS

–a Print all command names, even those containing unprintable characters and those used only once. By default, those are placed under the name '***other.'

–b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

–c Besides total user, system, and real time for each command print percentage of total time over all commands.

–d Sort by average number of disk I/O operations.

–D Print and sort by total number of disk I/O operations.

–f Force no interactive threshold compression with –v flag.

–i Do not read in summary file.

–j Instead of total minutes time for each category, give seconds per call.

–k Sort by CPU-time average memory usage.

–K Print and sort by CPU-storage integral.

–l Separate system and user time; normally they are combined.

–m Print number of processes and number of CPU minutes for each user.

–n Sort by number of calls.

–r Reverse order of sort.

–s Merge accounting file into summary file **/var/adm/savacct** when done.

–t For each command report ratio of real time to the sum of user and system times.

–u Superseding all other flags, print for each record in the accounting file the user ID and command name.

−v Followed by a number *n*, types the name of each command used *n* times or fewer. If *n* is not specified, it defaults to 1. Await a reply from the terminal; if it begins with **y**, add the command to the category '**junk**.' This is used to strip out garbage.

−S The following filename is used as the command summary file instead of **/var/adm/savacct**.

−U The following filename is used instead of **/var/adm/usracct** to accumulate the per-user statistics printed by the **−m** option.

FILES

/var/adm/pacct	raw accounting
/var/adm/savacct	summary by command
/var/adm/usracct	summary by user ID

SEE ALSO

acct(2V), **acct**(5), **ac**(8)

BUGS

sa's execution time increases linearly with the magnitude of the largest positive user ID in **/etc/passwd**.

NAME

savecore – save a core dump of the operating system

SYNOPSIS

/usr/etc/savecore [**–v**] *directory* [*system-name*]

DESCRIPTION

savecore saves a core dump of the kernel (assuming that one was made) and writes a reboot message in the shutdown log. It is meant to be called near the end of the **/etc/rc.local** file after the system boots. However, it is not normally run by default. You must edit that file to enable it.

savecore checks the core dump to be certain it corresponds with the version of the operating system currently running. If it does, **savecore** saves the core image in the file *directory*/**vmcore.***n* and the kernel's namelist in *directory*/**vmunix.***n*. The trailing .*n* in the pathnames is replaced by a number which grows every time **savecore** is run in that directory.

Before **savecore** writes out a core image, it reads a number from the file *directory*/**minfree**. This is the minimum number of kilobytes that must remain free on the filesystem containing *directory*. If there is less free space on the filesystem containing *directory* than the number of kilobytes specified in **minfree**, the core dump is not saved. If the **minfree** file does not exist, **savecore** always writes out the core file (assuming that a core dump was taken).

savecore also logs a reboot message using facility **LOG_AUTH** (see **syslog**(3)). If the system crashed as a result of a panic, **savecore** logs the panic string too.

If the core dump was from a system other than /**vmunix**, the name of that system must be supplied as *system-name*.

OPTIONS

–v Verbose. Enable verbose error messages from **savecore**.

FILES

directory/**vmcore.***n*
directory/**vmunix.***n*
directory/**minfree**
/vmunix the kernel
/etc/rc.local

SEE ALSO

syslog(3), **panic**(8S), **sa**(8)

BUGS

savecore can be fooled into thinking a core dump is the wrong size.

You must run **savecore** very soon after booting — before the swap space containing the crash dump is overwritten by programs currently running.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [**–ba**] [**–bd**] [**–bi**] [**–bm**] [**–bp**] [**–bs**] [**–bt**] [**–bv**] [**–bz**]
[**–C***file*] [**–d***X*] [**–F***fullname*] [**–f***name*] [**–h***N*] [**–n**] [**–o***x value*] [**–q**[*time*]]
[**–r***name*] [**–R***string*] [**–t**] [**–v**] [*address* ...]

DESCRIPTION

sendmail sends a message to one or more people, routing the message over whatever networks are necessary. **sendmail** does internetwork forwarding as necessary to deliver the message to the correct place.

sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; **sendmail** is used only to deliver pre-formatted messages.

With no flags, **sendmail** reads its standard input up to an EOF, or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in the local **aliases**(5) file, or by using the Network Interface Service (NIS), and aliased appropriately. In addition, if there is a **.forward** file in a recipient's home directory, **sendmail** forwards a copy of each message to the list of recipients that file contains. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

sendmail will also route mail directly to other known hosts in a local network. The list of hosts to which mail is directly sent is maintained in the file **/usr/lib/mailhosts**.

OPTIONS

–ba Go into ARPANET mode. All input lines must end with a LINEFEED, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

–bd Run as a daemon, waiting for incoming SMTP connections.

–bi Initialize the alias database.

–bm Deliver mail in the usual way (default).

–bp Print a summary of the mail queue.

–bs Use the SMTP protocol as described in RFC 821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.

–bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

–bv Verify names only — do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

–bz Create the configuration freeze file.

–C*file* Use alternate configuration file.

–d*X* Set debugging value to *X*.

–F*fullname* Set the full name of the sender.

–f*name* Sets the name of the "from" person (that is, the sender of the mail). **–f** can only be used by "trusted" users (who are listed in the config file).

–h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.

–M*id* Attempt to deliver the queued message with message-id **id**.

–n Do not do aliasing.

–o*x value* Set option *x* to the specified *value*. Options are described below.

–q[*time]* Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *time* is given as a tagged number, with **s** being seconds, **m** being minutes, **h** being hours, **d** being days, and **w** being weeks. For example, **–q1h30m** or **–q90m** would both set the timeout to one hour thirty minutes.

–r*name* An alternate and obsolete form of the **–f** flag.

–R*string* Go through the queue of pending mail and attempt to deliver any message with a recipient containing the specified string. This is useful for clearing out mail directed to a machine which has been down for awhile.

–t Read message for recipients. ''To:'', ''Cc:'', and ''Bcc:'' lines will be scanned for people to send to. The ''Bcc:'' line will be deleted before transmission. Any addresses in the argument list will be suppressed.

–v Go into verbose mode. Alias expansions will be announced, etc.

PROCESSING OPTIONS

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file* Use alternate alias file.

c On mailers that are considered ''expensive'' to connect to, do not initiate immediate connection. This requires queueing.

d*x* Set the delivery mode to **x**. Delivery modes are **i** for interactive (synchronous) delivery, **b** for background (asynchronous) delivery, and **q** for queue only — that is, actual delivery is done the next time the queue is run.

D Run **newaliases**(8) to automatically rebuild the alias database, if necessary.

e*x* Set error processing to mode **x**. Valid modes are **m** to mail back the error message, **w** to ''write'' back the error message (or mail it back if the sender is not logged in), **p** to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes **m** or **w** and if the sender is local to this machine, a copy of the message is appended to the file **dead.letter** in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-system-style ''From'' lines at the front of messages.

g*N* The default group ID to use when calling mailers.

H*file* The **SMTP** help file.

i Do not take dots on a line by themselves as a message terminator.

L*n* The log level.

m Send to ''me'' (the sender) also if I am in an alias expansion.

o If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

Q*queuedir*
Select the directory in which to queue messages.

r*timeout*
The timeout on reads; if none is set, **sendmail** will wait forever for a mailer.

S*file* Save statistics in the named file.

s Always instantiate the queue file, even under circumstances where it is not strictly necessary.

T*time* Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.

t*stz,dtz* Set the name of the time zone.

u*N* Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep **sendmail** from suppressing the blanks from between arguments.

sendmail returns an exit status describing what it did. The codes are defined in **sysexits.h**

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as **newaliases**, **sendmail** rebuilds the alias database. If invoked as **mailq**, **sendmail** prints the contents of the mail queue.

FILES

Except for **/etc/sendmail.cf**, these pathnames are all specified in **/etc/sendmail.cf**. Thus, these values are only approximations.

/etc/aliases	raw data for alias names
/etc/aliases.pag	data base of alias names
/etc/aliases.dir	
/usr/lib/mailhosts	list of hosts to which mail can be sent directly
/etc/sendmail.cf	configuration file
/etc/sendmail.fc	frozen configuration
/etc/sendmail.hf	help file
/etc/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/usr/bin/mail	to deliver local mail
/var/spool/mqueue/*	temp files and queued mail
~/.forward	list of recipients for forwarding messages

SEE ALSO

biff(1), **bin-mail**(1), **mail**(1), **aliases**(5) **newaliases**(8)

System and Network Administration

Su, Zaw-Sing, and Jon Postel, *The Domain Naming Convention for Internet User Applications*, RFC 819, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

Postel, Jon, *Simple Mail Transfer Protocol*, RFC 821, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

Crocker, Dave, *Standard for the Format of ARPA-Internet Text Messages*, RFC 822, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

set4, unset4, check4 – set, unset, and check the 4 megabyte process virtual address space limit flag in a Sun386i module

SYNOPSIS

set4 [**–d** *working_directory*] [– | *filename*] ...

unset4 *filename* ...

check4 *filename* ...

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

set4 sets the 4 megabyte process memory flag in each *filename* program image, limiting the virtual address space for each program to 4 megabytes. If a '–' is used, **set4** reads the standard input for a list of files to set the 4 megabyte limit on. Lines in the standard input whose first character is '#' are ignored, so files may include comments.

unset4 clears the 4 megabyte process memory flag in the program image, so the process virtual address space is not limited to 4 megabytes.

check4 reports programs that do not have the 4 megabyte limit set, and does not report programs with the limit set.

OPTIONS

–d *working_directory*
: This specifies a directory prefix for file names that **set4** processes.

EXAMPLES

Suppose that the file **small_progs** contains the following:

```
# These files should have their virtual address spaces limited to 4 MB:
/bin/date
/bin/true
```

Then the following command will run **set4** on **/build/bin/false**, **/build/bin/date**, **/build/bin/true**, and **/build/bin/cat**.

```
example% set4 –d /build /bin/false –
/bin/cat < small_progs
example%
```

In this example, **unset4** clears the 4 megabyte limit flag in **date**, and **clri**.

```
example% unset4 /bin/date /etc/clri
example%
```

In the last example, **check4** shows that **date** and **clri** are 4 megabyte processes, but **basename** is not.

```
example% check4 /bin/date /etc/clri /usr/bin/basename
basename is not a 4MB process
example%
```

SEE ALSO

execve(2V) **execl**(3V)

BUGS

There is a problem in the way that processes that have the 4 megabyte limit set **exec**( ) processes that do not have the limit set. (See **execve**(2V) and **execl**(3V) for descriptions of **exec**( ) processing.) For a short time during the **exec**( ), a child has the parent's data and stack limits. During this time, the program is checked to see if it will fit into memory. If the parent had the 4 megabyte limit set, the test fails, because the child program is running with the parent's 4 megabyte limit. This only affects programs which have more than 4 megabytes of global or static data compiled into the program. It does not affect programs which use **malloc**(3V) to obtain memory.

For example, **csh**(1) and **sh**(1) may be 4 megabyte processes. If they are, and if you try to run a program with more than 4 megabytes of global and static data, the shell cannot successfully **exec( )**. To fix this problem, become root on your machine and enter the following commands:

```
example% /etc/mount –o remount,rw /usr
/usr/etc/unset4 /bin/csh /bin/sh
example%
```

Then log out and back in again to run the modified shell. This makes **csh** and **sh** "normal" processes.

NAME

setsid – set process to session leader

SYNOPSIS

setsid [**–b**] *command* [*arguments*]

DESCRIPTION

setsid executes *command* after altering the execution environment such that the next non-controlling terminal opened will be assigned as *command*'s controlling terminal.

OPTIONS

–b Alteration to the execution environment persists across calls to **fork**(2V).

The **–b** option puts the process into a state that is supported in SunOS Release 4.1 solely as a migration aid; this option will not be supported in future releases.

EXAMPLES

Components of two SunLink products, **/usr/sunlink/dni/dnilogind** (the DECNET analog of **rlogind**(8C) and **/usr/sunlink/x25/x29** (the OSI analog of **rlogind**), are known to need this wrapper. Typical usage is:

```
example% cd /usr/sunlink/dni
example% mv dnilogind .dnilogind
example% cat > dnilogind
#!/bin/sh
/usr/etc/setsid –b /usr/sunlink/dni/.dnilogind "$@"
^D
example% chmod +x dnilogind
```

SEE ALSO

setsid(2V)

IEEE Std 1003.1-1988

NAME

showfh – print full pathname of file from the NFS file handle

SYNOPSIS

/usr/etc/showfh *server_name num1 num2 ... num8*

DESCRIPTION

showfh prints the full path name of the file on the server for the given file handle (*num1 ... num8*). *server_name* is the server from where the client got this file handle. *num1 ... num8* are the file handle numbers represented in hexadecimal notation.

The **showfhd** daemon should be running on the NFS servers to answer **showfh** requests. If it cannot find the file corresponding to the given file handle, it prints a diagnostic message.

SEE ALSO

showfhd(8C)

BUGS

If the given NFS file handle is stale, then **showfh** may not print the name of the actual file. The inode for the file could have been allocated to some other file.

NAME

showfhd – showfh daemon run on the NFS servers

SYNOPSIS

/usr/etc/rpc.showfhd

DESCRIPTION

showfhd is the daemon which runs on the NFS servers and answers **showfh** requests. It provides the full path name for the given file handle. If it cannot find the file for the corresponding inode number, it returns an error message.

FILES

/etc/mtab	table of mounted file systems

SEE ALSO

find(1), **showfh**(8C)

NAME

showmount – show all remote mounts

SYNOPSIS

/usr/etc/showmount [**–ade**] [*hostname*]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the **mountd**(8C) server on *host*, and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the value returned by **hostname**(1).

OPTIONS

–a Print all remote mounts in the format:

hostname:*directory*

where *hostname* is the name of the client, and **directory** is the root of the file system that has been mounted.

–d List directories that have been remotely mounted by clients.

–e Print the list of exported file systems.

FILES

/etc/rmtab

SEE ALSO

hostname(1), **exports**(5), **exports**(5), **exportfs**(8), **mountd**(8C)

BUGS

If a client crashes, its entry will not be removed from the list until it reboots and executes '**umount –a**'.

NAME

shutdown – close down the system at a given time

SYNOPSIS

/usr/etc/shutdown [**–fhknr**] [*time* [*warning-message* ...]

DESCRIPTION

shutdown provides an automated procedure to notify users when the system is to be shut down. *time* specifies when **shutdown** will bring the system down; it may be the word **now** (indicating an immediate shutdown), or it may specify a future time in one of two formats: +*number* and *hour*:*min*. The first form brings the system down in *number* minutes, and the second brings the system down at the time of day indicated in 24-hour notation.

At intervals that get closer as the apocalypse approaches, warning messages are displayed at terminals of all logged-in users, and of users who have remote mounts on that machine. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating **/etc/nologin** and writing a message there. If this file exists when a user attempts to log in, **login**(1) prints its contents and exits. The file is removed just before **shutdown** exits.

At shutdown time a message is written to the system log daemon, **syslogd**(8), containing the time of shutdown, the instigator of the shutdown, and the reason. Then a terminate signal is sent to **init**, which brings the system down to single-user mode.

The time of the shutdown and the warning message are placed in **/etc/nologin**, which should be used to inform the users as to when the system will be back up, and why it is going down (or anything else).

OPTIONS

As an alternative to the above procedure, these options can be specified:

–f Shut the system down in the manner of **fasthalt** (see **fastboot**(8)), so that when the system is rebooted, the file systems are not checked.

–h Execute **halt**(8).

–k Simulate shutdown of the system. Do not actually shut down the system.

–n Prevent the normal **sync**(2) before stopping.

–r Execute **reboot**(8).

FILES

/etc/nologin	tells login not to let anyone log in
/etc/xtab	list of remote hosts that have mounted this host

SEE ALSO

login(1), **sync**(2), **fastboot**(8), **halt**(8), **reboot**(8), **syslogd**(8)

BUGS

Only allows you to bring the system down between "now" and 23:59 if you use the absolute time for shutdown.

NAME

skyversion – print the SKYFFP board microcode version number

SYNOPSIS

/usr/etc/skyversion

DESCRIPTION

skyversion obtains from the SKYFFP board the Sky version number of the microcode currently loaded and prints the result on the standard output.

DIAGNOSTICS

The Sky version number operation code used to implement this command is not available for microcode releases earlier than Sky release 3.00. The result in this case is unpredictable and is either a nonmeaningful version number or a message indicating that no version number is available.

Meaningful version numbers are of the form *n.dd* where $n \geq 3$.

NAME

spray – spray packets

SYNOPSIS

/usr/etc/spray [**–c** *count*] [**–d** *delay*] [**–i** *delay*] [**–l** *length*] *host*

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

spray sends a one-way stream of packets to *host* using RPC, and reports how many were received, as well as the the transfer rate. The *host* argument can be either a name or an internet address.

OPTIONS

–c *count*
: Specify how many packets to send. The default value of *count* is the number of packets required to make the total stream size 100000 bytes.

–d *delay*
: Specify how many microseconds to pause between sending each packet. The default is 0.

–i *delay*
: Use ICMP echo packets rather than RPC. Since ICMP automatically echos, this creates a two way stream.

–l *length*
: The *length* parameter is the numbers of bytes in the Ethernet packet that holds the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32 bit quantities, not all values of *length* are possible, and **spray** rounds up to the nearest possible value. When *length* is greater than 1514, then the RPC call can no longer be encapsulated in one Ethernet packet, so the *length* field no longer has a simple correspondence to Ethernet packet size. The default value of *length* is 86 bytes (the size of the RPC and UDP headers).

SEE ALSO

icmp(4P), **ping**(8C), **sprayd**(8C)

Installing SunOS 4.1

NAME

sprayd, rpc.sprayd – spray server

SYNOPSIS

/usr/etc/rpc.sprayd

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

rpc.sprayd is a server which records the packets sent by **spray**(8C), and sends a response to the originator of the packets. The **rpc.sprayd** daemon is normally invoked by **inetd**(8C).

SEE ALSO

inetd(8C), **spray**(8C)

Installing SunOS 4.1

NAME

start_applic – generic application startup procedures

SYNOPSIS

/usr/etc/start_applic

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

start_applic is a short generic shell script that can be copied or symbolically linked into either **/vol/local/bin/***application* or **/usr/local/bin/***application*. When invoked as *application*, an application installed as described below will be correctly invoked on systems of any supported processor architecture. Installing **start_applic** (or a customized version of it) in one of these locations ensures that no user's or system's environment needs to be modified just to run the application. Applications are stored in a single tree which is not shared with any other applications. This tree may be available on different systems in different places; if the application needs to reference its distribution tree, this should be determined from the *application*_ROOT environment variable.

The application startup script arranges that the PATH and *application*_ROOT environment variables are set correctly while the application is running. If the application's distribution tree (placed into **/vol/***application* or **/usr/local/***application*) does not have an executable binary with the name of the application (for example, **/vol/***application***/bin.***arch***/***application*), then **start_applic** can not be used, and a customized application startup script must be used instead. Such scripts must also allow users to invoke the application from systems of any architecture, without requiring them to customize their own environments.

Note that there are two contrasting models of software installation. The **heterogeneous model** assumes general availability of the software, and solves the "which binaries to use" problem with no administrative overhead. The **homogeneous model** assumes very limited availability of software, requires administrative procedures to ensure that **/usr/local** only contains binaries of the local architecture, and does not really account for networked installations. It is easier to add support for additional architectures using a heterogeneous network model of software installation from the beginning.

Heterogeneous Networked Installations

Applications available on the network are available through **/vol/***application* and exported either to all systems or just to selected ones, as licensing restrictions allow. The export point is **/export/vol/***application*, which is a symbolic link to the actual installation point, typically the **/files/vol/***application* directory. All subdirectories not explicitly tagged with a processor architecture are shared among all processor architectures; thus while the **.../bin.sun386** and **.../lib.sun386** subdirectories contain, respectively, binaries and libraries executable only on systems of the Sun386i architecture, the **.../bin** directory contains executables that run on any architecture (typically using an interpreter such as **/bin/sh**), and the **.../etc** directory only contains sharable configuration files.

Homogeneous Single Machine Installations

Applications available only on a specific machine and its boot clients of the same architecture are installed into **/usr/local/***application*. This directory supports only a single architecture, so **/usr/local/***application***/bin** contains binaries executable only on the local architecture, and **/usr/local/***application***/lib** contains libraries executable only on the local architecture. Any sharable files are grouped in **/usr/local/***application***/share**.

To install an application onto a boot server to serve boot clients with other architectures, place the application in **/usr/local/***application* on the clients, as described above. The installation point (on the server) for application binaries of architecture *arch* is **/export/local/***arch***/***application*. When the architecture is the server architecture, this case is identical with the one above.

Other Installations

Smaller applications (of only one or two files) may be installed into the appropriate **/vol/local/bin.***arch* directory, or possibly into **/export/local/***arch***/bin**. These directories are in user's default paths, so the application does not need to be registered using **start_applic**.

FILES

/files<n>/vol/*application*
/export/vol/*application*
/vol/*application*
/vol/*application***/bin.***arch***/ application**
/usr/local/*application*
/export/local/*arch/application*

SEE ALSO

auto.vol(5), **exports**(5), **automount**(8), **exportfs**(8)

Sun386i SNAP Administration

Sun386i Advanced Administration

NAME

statd, rpc.statd – network status monitor

SYNOPSIS

/usr/etc/rpc.statd

DESCRIPTION

statd is an intermediate version of the status monitor. It interacts with **lockd**(8C) to provide the crash and recovery functions for the locking services on NFS.

FILES

/etc/sm
/etc/sm.bak
/etc/state

SEE ALSO

statmon(5), **lockd**(8C)

BUGS

The crash of a site is only detected upon its recovery.

NAME

sticky – mark files for special treatment

DESCRIPTION

The *sticky bit* (file mode bit 01000, see **chmod**(2V)) is used to indicate special treatment of certain files and directories. A directory for which the sticky bit is set restricts deletion of files it contains. A file in a sticky directory may only be removed or renamed by a user who has write permission on the directory, and either owns the file, owns the directory, or is the super-user. This is useful for directories such as **/tmp**, which must be publicly writable, but should deny users permission to arbitrarily delete or rename the files of others.

If the sticky bit is set on a regular file and no execute bits are set, the system's page cache will not be used to hold the file's data. This bit is normally set on swap files of diskless clients so that accesses to these files do not flush more valuable data from the system's cache. Moreover, by default such files are treated as swap files, whose inode modification times may not necessarily be correctly recorded on permanent storage.

Any user may create a sticky directory. See **chmod** for details about modifying file modes.

BUGS

mkdir(2V) will not create a file with the sticky bit set.

FILES

/tmp

SEE ALSO

chmod(1V), **chmod**(2V), **chown**(2V), **mkdir**(2V)

NAME

sundiag – system diagnostics

SYNOPSIS

/usr/diag/sundiag/sundiag [**–Cmt**] [**–k** *kernel_name*] [**–o** *saved_options_file*]
[*generic_tool_arguments*]

AVAILABILITY

This program is available with the *User Diagnostics* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

sundiag is a diagnostic facility that tests the functionality of the operating system and reports its findings. It can also be used to report the hardware configuration as detected by the system.

You must be **root** to use **sundiag**.

When run on the console monitor, **sundiag** takes full advantage of the *SunView 1* windowing environment. There are four subwindows:

- A control panel for displaying the discovered hardware configuration and manipulating of the numerous test parameters and options.
- A test status panel which shows the test results.
- A console window which is used to display messages.
- A performance monitor.

There are also some popup frames, including a text frame for viewing **sundiag** and system log files.

When executed from a terminal, **sundiag** uses **curses**(3V) to simulate each subwindow on the screen.

sundiag consists of **sundiag**, along with several binary modules and executable files containing the actual test code, all of which reside in **/usr/diag/sundiag**.

OPTIONS

–C Redirect the console output from any existing console window to the **sundiag** console subwindow.

–m Create a device file for all devices found during the kernel probe. **sundiag** uses the same major/minor device numbers and permissions declared in **/dev/MAKEDEV**.

–t Run **sundiag** on a terminal.

–k *kernel_name*
Specify the customized kernel name that was used to boot up the system. The default kernel name is **/vmunix**. Since the **rstatd**(8C) that the performance monitor requires is hard-wired to use **/vmunix** as the kernel name, the performance monitor is disabled when this option is specified.

–o *saved_options_file*
Use the *saved_options_file* to restore options. The default option file is **.sundiag**. **.sundiag** is used if the **–o** option is not used and if the default file exists.

generic_tool_arguments
Refer to **sunview**(1) for examples of generic tool arguments that may be used with **sundiag**.

FILES

/var/adm/sundiaglog/options/.sundiag	start-up option file
/usr/diag/sundiag/.usertest	user-defined test description file

SEE ALSO

sunview(1), **curses**(3V), **rstatd**(8C)

Installing SunOS 4.1
Sundiag User's Guide

NAME

suninstall – install and upgrade the SunOS operating system

SYNOPSIS

/usr/etc/install/suninstall

DESCRIPTION

suninstall is a forms-based subsystem for installing and upgrading the SunOS operating system. Unlike previous installation subsystems, **suninstall** does not require recapitulation of an interrupted procedure; you can pick up where you left off. A new invocation of **suninstall** displays the saved information and offers the user an opportunity to make any needed alterations before it proceeds.

Note: **suninstall** only exists in the mini-root and should only be invoked from there (see *Installing SunOS 4.1*).

suninstall allows installation of the operating system onto any system configuration, be it standalone, dataless, a homogeneous file server, or a heterogeneous server. It installs the various versions of the operating system needed by clients on a heterogeneous file server, from any Sun distribution media format. The number of different system versions that can be installed is only limited to the disk space available.

After the initial installation, the suninstall utility program **add_client**(8) adds clients while the server is running in multiuser mode. The suninstall **add_services**(8) program converts a standalone system or server into a heterogeneous file server, without rebooting, while the system is running in multiuser mode. To remove a diskless client, use the suninstall **rm_client**(8) program in multiuser mode.

To abort the installation procedure, use the interrupt character (typically CTRL-C).

USAGE

Refer to *Installing SunOS 4.1* for more information on the various menus and selections.

FILES

/usr/etc/install	directory containing installation programs and scripts
/usr/etc/install/xdrtoc	subsystem utility program
/etc/install	directory containing suninstall data files

SEE ALSO

add_client(8), **add_services**(8), **extract_unbundled**(8), **rm_client**(8)

Installing SunOS 4.1

NOTES

It is advisable to exit **suninstall** through the exit options from the **suninstall** menus.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/usr/etc/swapon −a

/usr/etc/swapon *name*...

DESCRIPTION

swapon specifies additional devices or files on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to **swapon** normally occur in the system multi-user initialization file **/etc/rc** making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

The second form gives individual block devices or files as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

Note: "swap files" made with **mkfile**(8) can be used as swap areas over NFS.

OPTIONS

−a Make available all devices of type **swap** in **/etc/fstab**. Using **swapon** with the **−a** option is the normal usage.

FILES

/dev/sd?b
/dev/xy?b
/dev/xd?b normal paging devices
/etc/fstab
/etc/rc

SEE ALSO

swapon(2), **fstab**(5), **init**(8), **mkfile**(8)

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sys-config – configure a system or administer configuration information

SYNOPSIS

/usr/etc/install/sys-config

DESCRIPTION

sys-config "unpacks" a machine and sets up its configuration. **sys-config** automatically runs when a pre-installed system is booted for the first time. It should not be run by hand. Instead, run **sys-unconfig**(8) to return the system to its pre-installed state. Then, reboot system, which will run **sys-config** automatically.

A system's configuration consists of hostname, Network Interface Service (NIS) domain name, timezone and IP address.

sys-config does the following:

- Edits the **/etc/hosts** with the correct hostname and IP address.
- Sets the hostname in **/etc/rc.boot**.
- Sets the domainname in **/etc/rc.single**.
- Sets the **/usr/lib/zoneinfo/localtime** file.
- Enables the Network Information Service (NIS) if the NIS service was requested.

When **sys-config** is finished, it prompts for a system reboot.

The default answer to any particular question is the current value of that configuration parameter. Parameters that have not changed can be quickly skipped over to get to the one that should be changed by typing a RETURN.

sys-config is potentially a dangerous utility and can be run only by the super-user.

FILES

/etc/hosts
/usr/lib/zoneinfo/localtime
/usr/etc/install/sys_info

SEE ALSO

sys-unconfig(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

sys-unconfig – undo a system's configuration

SYNOPSIS

/usr/etc/install/sys-unconfig

DESCRIPTION

sys-unconfig packs up a machine to make it ready to be configured again.

It restores a systems's configuration to an "as-manufactured" state. A system's configuration consists of hostname, Network Interface Service (NIS) domain name, timezone and IP address.

sys-unconfig does the following:

- Restores the default **/etc/hosts** file.
- Removes the default hostname in **/etc/hostname.??[0-9]**.
- Removes the default domainname in **/etc/defaultdomain.**
- Removes the default **/usr/lib/zoneinfo/localtime** file.
- Disables the Network Information Service (NIS) if the NIS service was requested.

When **sys-unconfig** is finished, it will prompt for a system shutdown.

sys-unconfig is potentially a dangerous utility and can only be run by the super-user.

FILES

/etc/hosts
/usr/lib/zoneinfo/localtime
/usr/etc/install/sys_info

SEE ALSO

sys-config(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

syslogd – log system messages

SYNOPSIS

/usr/etc/syslogd [**–d**] [**–f***configfile*] [**–m** *interval*]

DESCRIPTION

syslogd reads and forwards system messages to the appropriate log files and/or users, depending upon the priority of a message and the system facility from which it originates. The configuration file **/etc/syslog.conf** (see **syslog.conf**(5)) controls where messages are forwarded. **syslogd** logs a mark (timestamp) message every *interval* minutes (default 20) at priority **LOG_INFO** to the facility whose name is given as **mark** in the **syslog.conf** file.

A system message consists of a single line of text, which may be prefixed with a priority code number enclosed in angle-brackets (< >); priorities are defined in **sys/syslog.h.**

syslogd reads from the **AF_UNIX** address family socket **/dev/log**, from an Internet address family socket specified in **/etc/services**, and from the special device **/dev/klog** (for kernel messages).

syslogd reads the configuration file when it starts up, and again whenever it receives a HUP signal, at which time it also closes all files it has open, re-reads its configuration file, and then opens only the log files that are listed in that file. **syslogd** exits when it receives a **TERM** signal.

As it starts up, **syslogd** creates the file **/etc/syslog.pid**, if possible, containing its process ID (PID).

Sun386i DESCRIPTION

syslogd translates messages using the databases specified on an optional line in the **syslog.conf** as indicated with a **translate** entry.

The format of these databases is described in **translate**(5).

OPTIONS

–d	Turn on debugging.
–f*configfile*	Specify an alternate configuration file.
–m *interval*	Specify an interval, in minutes, between mark messages.

FILES

/etc/syslog.conf	configuration file
/etc/syslog.pid	process ID
/dev/log	AF_UNIX address family datagram log socket
/dev/klog	kernel log device
/etc/services	network services database

SEE ALSO

logger(1), **syslog**(3), **syslog.conf**(5), **translate**(5)

NAME

talkd, in.talkd – server for talk program

SYNOPSIS

/usr/etc/in.talkd

DESCRIPTION

talkd is a server used by the **talk**(1) program. It listens at the udp port indicated in the ''talk'' service description; see **services**(5). The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

SEE ALSO

talk(1), **services**(5), **inetd**(8C)

BUGS

The protocol is architecture dependent, and can not be relied upon to work between Sun systems and other machines.

NAME

telnetd, in.telnetd – TCP/IP TELNET protocol server

SYNOPSIS

/usr/etc/in.telnetd

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

telnetd is a server which supports the TCP/IP standard TELNET virtual terminal protocol. **telnetd** is invoked by the internet server (see **inetd**(8C)), normally for requests to connect to the TELNET port as indicated by the **/etc/services** file (see **services**(5)).

telnetd operates by allocating a pseudo-terminal device (see **pty**(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as its standard input, output, and error. **telnetd** manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, **telnetd** sends TELNET options to the client side indicating a willingness to do *remote echo* of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in "cooked" mode, and with **XTABS**, **ICRNL**, and **ONLCR** enabled (see **termio**(4)).

telnetd is willing to *do*: *echo*, *binary*, *suppress go ahead*, and *timing mark*. **telnetd** is willing to have the remote client *do*: *binary*, *terminal type*, and *suppress go ahead*.

SEE ALSO

telnet(1C)

Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification," RFC 854, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

BUGS

Some TELNET commands are only partially implemented.

The TELNET protocol allows for the exchange of the number of lines and columns on the user's terminal, but **telnetd** doesn't make use of them.

Because of bugs in the original 4.2 BSD **telnet**(1C), **telnetd** performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD **telnet**(1C).

Binary mode has no common interpretation except between similar operating systems

The terminal type name received from the remote client is converted to lower case.

The *packet* interface to the pseudo-terminal (see **pty**(4)) should be used for more intelligent flushing of input and output queues.

telnetd never sends TELNET *go ahead* commands.

telnetd can only support 64 pseudo-terminals.

NAME

tfsd – TFS daemon

SYNOPSIS

/usr/etc/tfsd

DESCRIPTION

tfsd is the daemon for the Translucent File Service (TFS). This daemon is started by **inetd**(8C) whenever a TFS request is made.

tfsd looks up a file by looking in the frontmost directory (see **tfs**(4S)). If the file is not found in this directory, **tfsd** follows the *searchlink* from the frontmost directory to the directory immediately behind it. **tfsd** continues to search for the file until one of the following conditions is met:

- The file is found in a directory.
- There are no more searchlinks to follow.
- A *whiteout* entry for the file is found.

The searchlinks and whiteout entries are specified in **.tfs_info** files.

FILES

.tfs_info	holds searchlink and whiteout entries

SEE ALSO

unwhiteout(1), lsw(1), **tfs**(4S), **mount_tfs**(8)

NAME

tftpd, in.tftpd – TCP/IP Trivial File Transfer Protocol server

SYNOPSIS

/usr/etc/in.tftpd [–s] [*homedir*]

Sun386i SYNOPSIS

/usr/etc/in.tftpd [–s] [**–p**] [*homedir*]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

tftpd is a server that supports the TCP/IP Trivial File Transfer Protocol (TFTP). This server is normally started by **inetd**(8C) and operates at the port indicated in the **tftp** Internet service description in the **/etc/inetd.conf** file; see **inetd.conf**(5) for details.

Before responding to a request, the server attempts to change its current directory to *homedir*; the default value is **/tftpboot**.

Sun386i DESCRIPTION

The **tftpd** daemon acts as described above, except that it will perform certain filename mapping operations unless instructed otherwise by the **–p** command line argument or when operating in a secure environment. This mapping affects only TFTP boot requests and will not affect requests for existing files.

The semantics of the changes are as follows. Only filenames of the format *ip-address* or *ip-address.arch,* where *ip-address* is the IP address in hex, and *arch* is the hosts's architecture (as returned by the **arch**(1) command), that do not correspond to files in **/tftpboot**, are mapped. If the address is known through a Network Interface Service (NIS) lookup, any file of the form **/tftpboot/ip-address*** (with or without a suffix) is returned. If there are multiple such files, any one may be returned. If the *ip-address* is unknown (that is if the **ipalloc (8C)** service says the name service does not know the address), the filename is mapped as follows: Names without the *arch* suffix are mapped into the name **pnp.SUN3**, and names with the suffix are mapped into **pnp.** *arch*. That file is returned if it exists.

OPTIONS

–s　Secure. When specified, the directory change must succeed; and the daemon also changes its root directory to *homedir*.

The use of **tftp** does not require an account or password on the remote system. Due to the lack of authentication information, **tftpd** will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note: this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling this service.

tftpd runs with the user ID (UID) and group ID (GID) set to **–2**, under the assumption that no files exist with that owner or group. However, nothing checks this assumption or enforces this restriction.

Sun386i OPTIONS

–p　Disable pnp entirely. Do not map filenames.

Sun386i FILES

/tftpboot/*　　filenames are IP addresses

SEE ALSO

tftp(1C) **inetd**(8C), **ipallocd**(8C), **netconfig**(8C)

Sollins, K.R., *The TFTP Protocol (Revision 2)*, RFC 783, Network Information Center, SRI International, Menlo Park, Calif., June 1981.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

Sun386i WARNINGS

A request for an *ip-address* from a Sun-4 can be satisfied by a file named *ip-address***.386** for compatibility with some early Sun-4 PROM monitors.

NAME

tic – terminfo compiler

SYNOPSIS

tic [–v[*n*]] [–**c**] *filename*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

tic compiles a **terminfo**(5V) source file into the compiled format. The results are placed in the directory **/usr/share/lib/terminfo.** The compiled format is used by the **curses**(3V) library.

Each entry in the file describes the capabilities of a particular terminal. When a **use=***entry* field is given in a terminal entry, **tic** reads in the binary (compiled) description of the indicated *entry* from **/usr/share/lib/terminfo** to duplicate the contents of that entry within the one being compiled. However, if an *entry* by that name is specified in *filename*, the entry in that source file is used first. Also, if a capability is defined in both entries, the definition in the current entry's source file is used.

If the environment variable **TERMINFO** is set, that directory is searched and written to instead of **/usr/share/lib/terminfo.**

OPTIONS

–**v**[*n*]
: Verbose. Display trace information on the standard error. The optional integer argument is a number from 1 to 10, inclusive, indicating the desired level of detail. If *n* is omitted, the default is 1.

–**c**
: Only check *filename* for errors. Errors in **use=** links are not detected.

FILES

/usr/share/lib/terminfo/?/* compiled terminal description data base

SEE ALSO

fork(2V), **curses**(3V), **curses**(3V), **malloc**(3V), **term**(5), **terminfo**(5V)

BUGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 1024 bytes.

When the –**c** option is used, duplicate terminal names will not be diagnosed; however, when –**c** is not used, they will be.

For backward compatibility, cancelled capabilities will not be marked as such within the terminfo binary unless the entry name has a '+' within it. Such terminal names are only used for inclusion with a **use=** field, and typically aren't used for actual terminal names.

DIAGNOSTICS

Most diagnostic messages produced by **tic** are preceded with the approximate line number and the name of the entry being processed.

mkdir *name* **returned bad status**
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a seek(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
Out of memory
Not enough free memory was available (**malloc**(3V) failed).

Can't open *filename*
The named file could not be opened or created.

Error in writing *filename*
The named file could not be written to.

Can't*tlink* **filename** *to* **filename**
A link failed.

Error in re-reading compiled *filename*
The compiled file could not be read back in.

Premature EOF
The current entry ended prematurely.

Backspaced off beginning of line
This error indicates something wrong happened within **tic**.

Unknown Capability – *filename*
The named invalid capability was found within the file.

Wrong type used for capability ...
For example, a string capability was given a numeric value.

Unknown token type
Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.
name: **bad term name**
Line *n*: **Illegal terminal name** – *name*
Terminal names must start with a letter or digit
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

name: **terminal name too long.**
An extremely long terminal name was found.

name: **terminal name too short.**
A one-letter name was found.

name **defined in more than one entry. Entry being used is** *name* .
An entry was found more than once.

Terminal name *name* **synonym for itself**
A name was listed twice in the list of synonyms.

At least one synonym should begin
At least one of the names of the terminal should begin with a letter.

Illegal character – **c**
The given invalid character was found in the input file.

Newline in middle of terminal name
The trailing comma was probably left off of the list of names.

Missing comma
A comma was missing.

Missing numeric value
The number was missing after a numeric capability.

NULL string value
The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
Self-explanatory.

Unknown option. Usage is:
An invalid option was entered.

Too many file names. Usage is:
Self-explanatory.

name **non-existent or permission denied**
The given directory could not be written into.

name **is not a directory**
Self-explanatory.

name: **Permission denied**
Access denied.

name: **Not a directory**
tic wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
A **fork**(2V) failed.

Error in following up use-links.
Either there is a loop in the links or they reference non-existent terminals. The following is a list of the entries involved:
A **terminfo**(5V) entry with a **use=***name* capability either referenced a non-existent terminal called *filename* or *filename* somehow referred back to the given entry.

NAME

tnamed, in.tnamed – TCP/IP Trivial name server

SYNOPSIS

/usr/etc/in.tnamed [−v]

DESCRIPTION

tnamed is a server that supports the TCP/IP Name Server Protocol. The name server operates at the port indicated in the "name" service description (see **services**(5)), and is invoked by **inetd**(8C) when a request is made to the name server.

Two known clients of this service are the MIT PC/IP software the Bridge boxes.

OPTIONS

−v   Invoke the daemon in verbose mode.

SEE ALSO

uucp(1C), **services**(5), **inetd**(8C)

Postel, Jon, *Internet Name Server*, IEN 116, SRI International, Menlo Park, California, August 1979.

BUGS

The protocol implemented by this program is obsolete. Its use should be phased out in favor of the Internet Domain protocol. See **named**(8C).

NAME

trpt – transliterate protocol trace

SYNOPSIS

/usr/etc/trpt [**–afjst**] [**–p***hex-address*] [*system* [*core*]]

DESCRIPTION

trpt interrogates the buffer of TCP trace records created when a socket is marked for "debugging" (see **getsockopt**(2)), and prints a readable description of these records. When no options are supplied, **trpt** prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

OPTIONS

–a In addition to the normal output, print the values of the source and destination addresses for each packet recorded.

–f Follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached.

–j Just give a list of the protocol control block addresses for which there are trace records.

–s In addition to the normal output, print a detailed description of the packet sequencing information.

–t In addition to the normal output, print the values for all timers at each point in the trace.

–p *hex-address*
Show only trace records associated with the protocol control block, the address of which follows.

The recommended use of **trpt** is as follows. Isolate the problem and enable debugging on the **socket**(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to **netstat**(8C). Then run **trpt** with the **–p** option, supplying the associated protocol control block addresses. The **–f** option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

getsockopt(2), **netstat**(8C)

DIAGNOSTICS

no namelist When the system image does not contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable and should be described here.

NAME

ttysoftcar – enable/disable carrier detect

SYNOPSIS

ttysoftcar [**−y** | **−n**] *tty* . . .

ttysoftcar −a

DESCRIPTION

For each *tty* specified **ttysoftcar** changes the carrier detect flag using the **TIOCSSOFTCAR ioctl( )** request (see **tty**(4)). If the **−a** option is specified, **ttysoftcar** sets all **tty**'s in the **/etc/ttytab** file to the carrier detection mode specified by their status field. If this field is set to **local**, software carrier detection is turned on. If this field is set to anything other than **local**, as is usually the case for modems, software carrier detection is turned off. **ttysoftcar** ignores devices in the **/etc/ttytab** file which do not exist.

If no options are specified, **ttysoftcar** returns the current status for *tty*. This status is reported as **y** or **n**.

OPTIONS

−a Reset **ttys** to appropriate values based on the status field of the **/etc/ttytab** file.

−y Turn on software carrier detect.

−n Turn off software carrier detect. Use hardware carrier detect.

SEE ALSO

termio(4), zs(4S), **ttytab**(5)

NAME

tunefs – tune up an existing file system

SYNOPSIS

/usr/etc/tunefs [**–a** *maxcontig*] [**–d** *rotdelay*] [**–e** *maxbpg*] [**–m** *minfree*] *special* | *filesystem*

DESCRIPTION

tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the OPTIONS given below:

OPTIONS

–a *maxcontig*
This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see **–d** below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d *rotdelay*
This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e *maxbpg*
This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

–m *minfree*
This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note: if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), **dumpfs**(8), **mkfs**(8), **newfs**(8)

System and Network Administration

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems; if run on the root file system, the system must be rebooted.

NAME

tzsetup – set up old-style time zone information in the kernel

SYNOPSIS

/usr/etc/tzsetup

DESCRIPTION

tzsetup attempts to find the offset from GMT and old-style Daylight Savings Time correction type (see **gettimeofday**(2)) that most closely matches the default time zone for the machine, and to pass this information to the kernel with a **settimeofday** ( ) call (see **gettimeofday**(2)). This is necessary if programs built under releases of SunOS prior to 4.0 are to be run; those programs get time zone information from the kernel using **gettimeofday**.

If it cannot find the offset from GMT, the offset is set to 0; if it cannot find the Daylight Savings Time correction type, it is set to **DST_NONE**, indicating that no Daylight Savings Time correction is to be performed.

DIAGNOSTICS

tzsetup: Can't open /usr/share/lib/zoneinfo/localtime: *reason*
The time zone file for the current time zone could not be opened.

tzsetup: Error reading /usr/lib/zoneinfo/localtime: *reason*
The time zone file for the current time zone could not be read.

tzsetup: Two or more time zone types are equally valid — no DST selected
There were two or more Daylight Savings Time correction types that generated results that were equally close to the correct results. None of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

tzsetup: No old-style time zone type is valid — no DST selected
None of the Daylight Savings Time correction types generated results that were in any way correct; none of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

tzsetup: Warning: No old-style time zone type is completely valid
None of the Daylight Savings Time correction types generated results that were completely correct; the best of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

tzsetup: Can't set time zone
tzsetup was run by a user other than the super-user; only the super-user may change the kernel's notion of the current time zone.

SEE ALSO

gettimeofday(2), **tzfile**(5), **zic**(8)

NAME

uid_allocd, gid_allocd – UID and GID allocator daemons

SYNOPSIS

/usr/etc/rpc.uid_allocd
/usr/etc/rpc.gid_allocd

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The UID (or GID) allocator will temporarily allocate an unused UID (or GID) for use by account administration tools. It maintains a cache of UIDs (GIDs) that have been allocated by potentially multiple tools (or instances of tools) in a distributed system, so that they can create accounts (or groups) concurrently. It also provides the ability to safely enter a UID (GID) into the cache which was allocated using some other method, such as manually by an administrator; and the ability to delete entries from the cache. Entries in this cache persist for at least an hour even through system crashes.

These allocators are available on the system which contains the master copy of the list of UIDs (or GID). Since this list is currently maintained using the Network Interface Service (NIS), the service is available on the master of the **passwd.byuid (group.bygid)** NIS map. The service could be provided using a UID database service other than the NIS service.

This implementation uses DES authentication (the Sun Secure RPC protocol) to restrict access to this function. The only clients privileged to allocate UIDs (GIDs) are those whose net IDs are in the *accounts* group (fixed at GID 11). All machine IDs are allowed to allocate UIDs (GIDs).

If the file **/etc/ugid_alloc.range** exists, the allocator only allocates UIDs (GIDs) in the range listed there. This feature is intended to be used by sites which have multiple NIS domains on their networks; each NIS domain would be assigned a unique range of UIDs (GIDs). If the file exists, and the local NIS domain is not explicitly assigned a unique range of UIDs or GID, none will be allocated. Without a mechanism to ensure that UIDs are uniquely assigned between NIS domains that share resources, normal NFS security mechanisms (excluding Secure NFS) may fail to serve as an advisory security mechanism. Common alternative methods for ensuring UID uniqueness include using a function of some preexisting identifier such as an employee number, or using a single NIS domain for the entire site.

FILES

/var/yp/*domainname***/passwd.byuid.{dir,pag}**
/var/yp/*domainname***/group.bygid.{dir,pag}**
/var/yp/*domainname***/netid.byname.{dir,pag}**
/etc/uid_alloc.cache
/etc/gid_alloc.cache
/etc/ugid_alloc.range
/usr/include/rpcsvc/uid_alloc.x
/usr/include/rpcsvc/gid_alloc.x

SEE ALSO

snap(1), **ugid_alloc.range**(5), **logintool**(8)

BUGS

Using UID (GID) ranges does not solve the problem that two different machines, or groups of machines, may assign different meaning to a given UID (GID).

The current implementation of the daemon is tuned towards small lists of active UIDs (GIDs), both in the NIS service and in the cache it maintains.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

unadv – unadvertise a Remote File Sharing resource

SYNOPSIS

unadv *resource*

AVAILABILITY

This program is available with the *RFS* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

unadv unadvertises a Remote File Sharing (RFS) *resource*, which is the advertised symbolic name of a local directory, by removing it from the advertised information on the domain name server. **unadv** prevents subsequent remote mounts of that resource. It does not affect continued access through existing remote or local mounts.

An administrator at a server can unadvertise only those resources that physically reside on the local machine. A domain administrator can unadvertise any resource in the domain from the primary name server by specifying *resource* name as *domain.resource*. A domain administrator should only unadvertise another hosts resources to clean up the domain advertise table when that host goes down. Unadvertising another host's resource changes the domain advertise table, but not the host advertise table.

This command is restricted to the super-user.

If *resource* is not found in the advertised information, an error message will be sent to standard error.

SEE ALSO

adv(8), **fumount**(8), **nsquery**(8)

NAME

unconfigure – reset the network configuration for a Sun386i system

SYNOPSIS

/usr/etc/unconfigure [**–y**]

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

unconfigure restores most of the system configuration and status files to the state they were in when delivered by Sun Microsystems, Inc. It also deletes all user accounts (including home directories), Network Interface Service (NIS) information, and any diskless client configurations that were set up.

After running **unconfigure,** a system halts. Rebooting it to multi-user mode at this point will start automatic system installation.

unconfigure is intended for use in the following situations:

- As one of the final steps in Software Manufacturing.
- In systems being set up with temporary configurations, holding no user accounts or diskless clients. These will occur during demonstrations and evaluation trials.
- To allow systems that had been used as standalones to be upgraded to join a network in a role other than as a master server. (See instructions later.)

unconfigure is potentially a dangerous utility; it does not work unless invoked by the super-user. As a warning, unless the **–y** option is passed, it will require confirmation that all user files and system software configuration information is to be deleted.

This utility is *not* recommended for routine use of any sort.

Resetting Temporary Configurations

If users need to set up and tear down configurations, **unconfigure** can be used to restore the system to an essentially as-manufactured state. The main concern here is that user accounts will be deleted, so this should not be done casually.

To reset a temporary configuration, just become the super-user and invoke **unconfigure.**

Upgrading Standalones to Network Clients

Systems that are going to be networked should be networked from the very first, if at all possible. This eliminates whole classes of compatibility problems, such as pathnames and (in particular) user account clashes.

Automatic system installation directly supports upgrading a single standalone system to an NIS master, and joining any number of unused systems (or systems upon which **unconfigure** has been run) into a network.

However, in the situation where standalone systems that have been used extensively are to be joined to a network, **unconfigure** can be used in conjunction with automatic system installation by a knowledgeable super-user to change a system's configuration from standalone to network client. This procedure is not recommended for use by inexperienced administrators.

The following procedure is not needed unless user accounts or other data need to be preserved; it is intended to ensure that every UID and GID is changed so as not to clash with those in use on the network. It must be applied to each system that is being upgraded from a standalone to a network client.

The procedure is as follows:

- Identify all accounts and files that you will want to save. If there are none, just run **unconfigure** and install the system on the network. Do not follow the remaining steps.
- Copy **/etc/passwd** to **/etc/passwd.bak.**

- Rename all the files (including home directories) so that they aren't deleted. (See **FILES** below.) These will probably be only in **/export/home**.
- Run **unconfigure** and install the system on the network.
- For each account listed in **/etc/passwd.bak** that you want to save, follow this procedure:
  - Create a new account on the network; if the UID and GID are the same as in **/etc/passwd.bak** on the standalone, then skip the next step. However, be sure that you do not make two different accounts with the same UID.
  - Use the '**chown −R**' command to change the ownership of the home directories.
  - You may need to rename the files you just chowned above, for example to ensure that they are the user's home directory. This may involve updating the **auto.home**(5) and **auto.home**(5) NIS maps, as well.
- Delete **/etc/passwd.bak**.

FILES

unconfigure deletes the following files, if they are present, replacing some of them with the distribution version if one is supposed to exist:

/etc/.rootkey	**/etc/ethers**	**/etc/localtime**	**/etc/publickey**
/etc/auto.home	**/etc/exports**	**/etc/net.conf**	**/etc/sendmail.cf**
/etc/auto.vol	**/etc/fstab**	**/etc/netmasks**	**/etc/syslog.conf**
/etc/bootparams	**/etc/group**	**/etc/networks**	**/etc/systems**
/etc/bootservers	**/etc/hosts**	**/etc/passwd**	**/single/ifconfig**
/var/sysex/*			

and all files in **/var/yp** except those distributed with the operating system.

unconfigure truncates all files in **/var/adm**. All user home directories in **/export/home** are deleted, except those for the default user account **users**, which is shipped with the operating system. All diskless client configuration information stored in **/export/roots**, **/export/swaps**, and **/export/dumps** is deleted.

SEE ALSO

chgrp(1), **find**(1), **group**(5), **passwd**(5) **adduser**(8), **chown**(8)

BUGS

More of the system configuration files should be reset.

This does not yet support taking a workstation off the network temporarily, for example, to take it home over the weekend for use as a standalone, or to move it to another network while traveling. This should be the default behavior.

The procedure for upgrading standalones to network clients should be automated; currently, only upgrading a standalone to a master server is automated.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

NAME

update – periodically update the super block

SYNOPSIS

/usr/etc/update

DESCRIPTION

update is a program that executes the **sync**(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(1), **sync**(2), **init**(8)

NAME

user_agentd – user agent daemon

SYNOPSIS

/usr/etc/rpc.user_agentd

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

rpc.user_agentd is the remote service used by **snap**(1) to create, move, or delete home directories, and by the New User Accounts feature of **logintool**(8) to create new home directories. The user_agent daemon is normally invoked by **inetd**(8C), and runs on all non-diskless systems.

When creating a new home directory, the user_agent daemon executes the **copy_home**(8) script which resides in the home directory of the primary group to which a new user will be added.

SEE ALSO

snap(1), **copy_home**(8), **inetd**(8C), **logintool**(8)

NAME

uucheck – check the UUCP directories and Permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**–v**] [**–x** *debug_level*]

AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

uucheck checks for the presence of the UUCP system required files and directories. It also checks for some obvious errors in the **Permissions** file (**/usr/lib/uucp/Permissions**).

Note: **uucheck** can only be used by the super-user or **uucp**.

OPTIONS

–v Give a detailed explanation of how the UUCP programs will interpret the **Permissions** file.

–x *debug_level*
Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

FILES

/etc/uucp/Systems
/etc/uucp/Permissions
/etc/uucp/Devices
/etc/uucp/Maxuuscheds
/etc/uucp/Maxuuxqts
/var/spool/uucp/*
/var/spool/locks/LCK*
/var/spool/uucppublic/*

SEE ALSO

uucp(1C), **uustat**(1C), **uux**(1C), **uucico**(8C), **uusched**(8C)

BUGS

The program does not check file/directory modes or some errors in the **Permissions** file such as duplicate login or machine name.

NAME

uucico – file transport program for the UUCP system

SYNOPSIS

/usr/lib/uucp/uucico [**–r** *role_number*] [**–x** *debug_level*] [**–i** *interface*] [**–d** *spool_directory*]
–s *system_name*

AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

uucico is the file transport program for UUCP work file transfers. **uux**(1C) and **uucp**(1C) both queue jobs that will be transferred by **uucico**. It is normally started by the scheduler, **uusched (8C),** but can be started manually; this is done for debugging. For example, the script **Uutry** starts **uucico** with debugging turned on.

OPTIONS

–r *role_number*
: Specify the role that **uucico** should perform. *role_number* is the digit 1 for master mode or 0 for slave mode (default). Master mode should be specified when **uucico** is started by a program or **cron**(8).

–x *debug_level*
: Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

–i *interface*
: Define the interface used with **uucico**. This interface only affects slave mode. Known interfaces are UNIX (default).

FILES

/etc/uucp/Systems
/etc/uucp/Permissions
/etc/uucp/Devices
/etc/uucp/Devconfig
/etc/uucp/Sysfiles
/etc/uucp/Maxuuxqts
/etc/uucp/Maxuuscheds
/var/spool/uucp/*
/var/spool/locks/LCK*
/var/spool/uucppublic/*

SEE ALSO

uucp(1C), **uustat**(1C), **uux**(1C), **cron**(8), **uusched**(8C)

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**–m**] [**–d***directory*] [**–n***time*] [**–p***pre*]

DESCRIPTION

uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

OPTIONS

–d*directory*
: Clean the indicated spool directory.

–m
: Send mail to the owner of the file when it is deleted.

–n*time*
: Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).

–p*pre*
: Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following deletes all files older than the specified time.

uuclean will typically be started by **cron**(8).

FILES

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

SEE ALSO

uucp(1C), **uux**(1C), **cron**(8)

NAME

uucleanup – UUCP spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**–C***time*] [**–D***time*] [**–m***string*] [**–o***time*] [**–s***system*] [**–W***time*]
[**–x** *debug_level*] [**–X***time*]

AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

uucleanup will scan the spool directories for old files and take appropriate action to remove them in a useful way:

- Inform the requestor of send/receive requests for systems that cannot be reached.
- Return mail, which cannot be delivered, to the sender.
- Delete or execute **rnews** for **rnews** type files (depending on where the news originated — locally or remotely).
- Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1 day). Note: **uucleanup** will process as if all option *time*s were specified to the default values unless *time* is specifically set.

This program is typically started by the shell **uudemon.cleanup**, which should be started by **cron**(8).

OPTIONS

–C*time* Remove any **C.** files that are at least *time* days old (default 7 days), and send appropriate information to the requestor.

–D*time* Remove any **D.** files that are at least *time* days old (default 7 days), and make an attempt to deliver mail messages and execute **rnews** when appropriate.

–m*string*
Include this line in the warning message generated by the **–W** option. The default line is '**See your local administrator to locate the problem**'.

–o*time* Delete other files that are more than *time* days old (default 2 days).

–s*system*
Execute for the spool directory for the remote system *system* only.

–W*time*
Send a mail message to be sent to the requestor warning about the delay in contacting the remote for any **C.** files that are *time* days old (default 1 day). The message includes the *JOBID*, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**–m** option).

–x *debug_level*
Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

–X*time* Remove any **X.** files that are at least *time* days old (default 2 days). The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by **D.** processing.

FILES

/usr/lib/uucp	directory with commands used by **uucleanup** internally
/var/spool/uucp	spool directory

SEE ALSO

uucp(1C), **uux**(1C), **cron**(8)

NAME

uucpd – UUCP server

SYNOPSIS

/usr/etc/in.uucpd

AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

uucpd is the server for supporting UUCP connections over networks.

uucpd is invoked by **inetd**(8C) when a UUCP connection is established (that is, a connection to the port indicated in the "uucp" service specification; see **services**(5)), and executes the following protocol:

1) The server prompts with **login:**. The **uucico**(8C) process at the other end must supply a username.

2) Unless the username refers to an account without a password, the server then prompts with **Password:**. The **uucico** process at the other end must supply the password for that account.

If the username is not valid or is valid but refers to an account that does not have **/usr/lib/uucp/uucico** as its login shell, or if the password is not the correct password for that account, the connection is dropped. Otherwise, **uucico** is run, with the user ID, group ID, group set, and home directory for that account, with the environment variables **USER** and **LOGNAME** set to the specified username, and with a **–u** flag specifying the username. Entries are made in **/var/adm/wtmp** and **/var/adm/lastlog** for the username.

FILES

/var/adm/wtmp	accounting
/var/adm/lastlog	time of last login

SEE ALSO

services(5), **inetd**(8C), **uucico**(8C)

DIAGNOSTICS

All diagnostic messages are returned on the connection, after which the connection is closed.

user read
An error occurred while reading the username.

passwd read
An error occurred while reading the password.

Login incorect.
The username is invalid or refers to an account with a login shell other than **/usr/lib/uucp/uucico**, or the password is not the correct password for the account.

NAME
uusched – the scheduler for the UUCP file transport program

SYNOPSIS
/usr/lib/uucp/uusched [**–u** *debug_level*] [–x *debug_level*]

AVAILABILITY
This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION
uusched is the UUCP file transport scheduler. It is usually started by the daemon **uudemon.hour** that is started by **cron**(8) from an entry in the system **crontab** file:

39 * * * * /bin/su uucp –c "/usr/lib/uucp/uudemon.hour > /dev/null"

OPTIONS
–u *debug_level*
: Pass *debug_level* as '–x *debug_level*' to any invocations of **uucico**(8C) started by **uusched**.

–x *debug_level*
: Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

FILES
/etc/uucp/Systems
/etc/uucp/Permissions
/etc/uucp/Devices
/var/spool/uucp/*
/var/spool/locks/LCK*
/var/spool/uucppublic/*

SEE ALSO
uucp(1C), **uustat**(1C), **uux**(1C), **cron**(8), **uucico**(8C)

NAME

uuxqt – execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**–x** *debug_level*]

DESCRIPTION

uuxqt is the program that executes remote job requests from remote systems generated by the use of the **uux**(1C) command. **mail**(1) uses **uux** for remote mail requests. **uuxqt** searches the spool directories looking for **X.** files. For each **X.** file, **uuxqt** checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The **Permissions** file is used to validate file accessibility and command execution permission.

OPTIONS

–x *debug_level*

Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

ENVIRONMENT

There are two environment variables that are set before the **uuxqt** command is executed:

UU_MACHINE　Machine that sent the job (the previous one).

UU_USER　User that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

FILES

/etc/uucp/Permissions
/etc/uucp/Maxuuxqts
/var/spool/uucp/*
/var/spool/locks/LCK*

SEE ALSO

mail(1), **uucp**(1C), **uustat**(1C), **uux**(1C), **uucico**(8C)

NAME

vipw – edit the password file

SYNOPSIS

/usr/etc/vipw

DESCRIPTION

vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor.

vipw performs a number of consistency checks on the password entry for root, and will not allow a password file with a "mangled" root entry to be installed. It also checks the **/etc/shells** file to verify the login shell for root.

FILES

/etc/ptmp
/etc/shells

SEE ALSO

passwd(1), vi(1), **passwd**(5), **adduser**(8)

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–cfisS**] [*interval* [*count*]]

DESCRIPTION

vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and CPU activity.

Without options, **vmstat** displays a one-line summary of the virtual memory activity since the system has been booted. If *interval* is specified, **vmstat** summarizes activity over the last *interval* seconds. If a *count* is given, the statistics are repeated *count* times.

For example, the following command displays a summary of what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system.

```
example% vmstat 5

procs   memory               page     faults
r b w  avm  fre  re at  pi  po  fr  de  sr x0 x1 x2 x3  in  sy  cs  us sy id
2 0 0  918  286  0 0  0  0  0  0  0 1 0 0 0  4 12  5 3  5 91
1 0 0  846  254  0 0  0  0  0  0  0 6 0 1 0 42 153 31 7 40 54
1 0 0  840  268  0 0  0  0  0  0  0 5 0 0 0 27 103 25 8 26 66
1 0 0  620  312  0 0  0  0  0  0  0 6 0 0 0 26  76 25 6 27 67

CTRL-C
example%
```

The fields of **vmstat**'s display are:

procs Report the number of processes in each of the three following states:

- **r** in run queue
- **b** blocked for resources (i/o, paging, etc.)
- **w** runnable or short sleeper (< 20 secs) but swapped

memory Report on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.

- **avm** number of active virtual Kbytes
- **fre** size of the free list in Kbytes

page Report information about page faults and paging activity. The information on each of the following activities is averaged each five seconds, and given in units per second.

- **re** page reclaims — but see the –S option for how this field is modified.
- **at** number of attaches — but see the –S option for how this field is modified.
- **pi** kilobytes per second paged in
- **po** kilobytes per second paged out
- **fr** kilobytes freed per second
- **de** anticipated short term memory shortfall in Kbytes
- **sr** pages scanned by clock algorithm, per-second

disk Report number of disk operations per second (this field is system dependent). For Sun systems, four slots are available for up to four drives: "x0" (or "s0" for SCSI disks), "x1", "x2", and "x3".

faults Report trap/interrupt rate averages per second over last 5 seconds.

- **in** (non clock) device interrupts per second
- **sy** system calls per second
- **cs** CPU context switch rate (switches/sec)

cpu Give a breakdown of percentage usage of CPU time.

- **us** user time for normal and low priority processes
- **sy** system time
- **id** CPU idle

OPTIONS

−c Report cache flushing statistics. By default, report the total number of each kind of cache flushed since boot time. The types are: user, context, region, segment, page, and partial-page.

−f Report on the number of **forks** and **vforks** since system startup and the number of pages of virtual memory involved in each kind of fork.

−i Report the number of interrupts per device. Autovectored interrupts (including the clock) are listed first.

−s Display the contents of the **sum** structure, giving the total number of several kinds of paging-related events which have occurred since boot.

−S Report on swapping rather than paging activity. This option will change two fields in **vmstat**'s ''paging'' display: rather than the ''re'' and ''at'' fields, **vmstat** will report ''si'' (swap-ins), and ''so'' (swap-outs).

FILES

/dev/kmem
/vmunix

BUGS

If more than one autovectored device has the same name, interrupts are counted for all like-named devices regardless of unit number. Such devices are listed with a unit number of '**?**'.

NAME

ypbatchupd – NIS batch update daemon

SYNOPSIS

/usr/etc/rpc.ypbatchupd

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ypbatchupd(8C) is the remote service used by **snap**(1) and **logintool**(8) to update the Network Interface Service (NIS) database on the master server, and to push all modified NIS maps to NIS servers. It is normally started by **/etc/rc.local**.

SEE ALSO

snap(1), **logintool**(8), **rc**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypinit – build and install NIS database

SYNOPSIS

/usr/etc/yp/ypinit −m

/usr/etc/yp/ypinit –s *master_name*

DESCRIPTION

ypinit sets up a Network Interface Service (NIS) database on an NIS server. It can be used to set up a master or a slave server. You must be the super-user to run it. It asks a few, self-explanatory questions, and reports success or failure to the terminal.

It sets up a master server using the simple model in which that server is master to all maps in the data base. This is the way to bootstrap the NIS system; later if you want you can change the association of maps to masters.

Note: If there are both 3.x and 4.x NIS servers running in the network, the 4.x server should be configured as the master.

All databases are built from scratch, either from information available to the program at runtime, or from the ASCII data base files in **/etc**. These files are listed below under FILES. All such files should be in their "traditional" form, rather than the abbreviated form used on client machines.

An NIS database on a slave server is set up by copying an existing database from a running server. The *master_name* argument should be the hostname of an NIS server (either the master server for all the maps, or a server on which the data base is up-to-date and stable).

Read **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

OPTIONS

−m Indicate that the local host is to be the NIS master.

−s Set up a slave database.

FILES

/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/services
/etc/protocols
/etc/ethers

SEE ALSO

ypfiles(5), **makedbm**(8), **ypmake**(8), **yppush**(8), **ypserv**(8), **ypxfr**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypmake – rebuild NIS database

SYNOPSIS

cd /var/yp ; make [*map*]

DESCRIPTION

The file called **Makefile** in **/var/yp** is used by **make**(1) to build the Network Interface Service (NIS) database. With no arguments, **make** creates **dbm** databases for any NIS maps that are out-of-date, and then executes **yppush**(8) to notify slave databases that there has been a change.

If you supply a *map* on the command line, **make** will update that map only. Typing **make passwd** will create and **yppush** the password database (assuming it is out of date). Likewise, **make hosts** and **make networks** will create and **yppush** the host and network files, **/etc/hosts** and **/etc/networks**.

There are three special variables used by **make**: DIR, which gives the directory of the source files; NOPUSH, which when non-null inhibits doing a **yppush** of the new database files; and DOM, used to construct a domain other than the master's default domain. The default for DIR is **/etc**, and the default for NOPUSH is the null string.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

FILES

/var/yp
/etc/hosts
/etc/networks

SEE ALSO

make(1), **ypfiles**(5), **makedbm**(8), **yppush**(8), **ypserv**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yppasswdd, rpc.yppasswdd – server for modifying NIS password file

SYNOPSIS

/usr/etc/rpc.yppasswdd *filename* [*adjunct_file*] [**–nogecos**] [**–noshell**] [**–nopw**]
[**–m** *argument1 argument2* ...]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

yppasswdd is a server that handles password change requests from **yppasswd**(1). Unless an *adjunct_file* is specified, it changes a password entry in *filename*, which is assumed to be in the format of **passwd**(5). *filename* is the password file that provides the basis for the *passwd.byname* and *passwd.byuid* maps. This should not be confused with the servers */etc/passwd* file which controls access to the server. In particular this file should not contain an entry for the super user.

If an *adjunct_file* is specified or **/etc/security/passwd.adjunct** exists, this file will be changed instead of the *filename*. An entry in *filename* or *adjunct_file* will only be changed if the password presented by **yppasswd**(1) matches the encrypted password of that entry.

If the **–noshell –nogecos** or **–nopw** options are given then these fields may not be changed remotely using **chfn**, **chsh**, or **passwd**(1).

If the **–m** option is given, then after *filename* or *adjunct_file* is modified, a **make**(1) will be performed in */var/yp*. Any arguments following the flag will be passed to *make*.

This server is not run by default, nor can it be started up from **inetd**(8C). If it is desired to enable remote password updating for the Network Interface Service (NIS), then an entry for **yppasswdd** should be put in the **/etc/rc** file of the host serving as the master for the NIS **passwd** file.

EXAMPLE

If the NIS password file is stored as **/var/yp/passwd**, then to have password changes propagated immediately, the server should be invoked as

/usr/etc/rpc.yppasswdd /var/yp/passwd –m passwd DIR=/var/yp

FILES

/var/yp/Makefile
/etc/security/passwd.adjunct
/etc/rc

SEE ALSO

make(1), **yppasswd**(1), **passwd**(1), **passwd**(5), **passwd.adjunct**(5), **ypfiles**(5), **inetd**(8C), **ypmake**(8)

NOTES

The password file specified to *rpc.yppasswdd* may not be a link.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yppoll – version of NIS map at NIS server

SYNOPSIS

/usr/etc/yp/yppoll [**–h** *host*] [**–d** *domain*] *mapname*

DESCRIPTION

yppoll asks a **ypserv**(8) process what the order number is, and which host is the Network Interface Service (NIS) master server for the named map. If the server is a v.1 NIS protocol server, **yppoll** uses the older protocol to communicate with it. In this case, it also uses the older diagnostic messages in case of failure.

OPTIONS

–h *host* Ask the **ypserv** process at *host* about the map parameters. If *host* is not specified, the NIS server for the local host is used. That is, the default host is the one returned by **ypwhich**(8).

–d *domain*
Use *domain* instead of the default domain.

SEE ALSO

ypfiles(5), **ypserv**(8), **ypwhich**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

yppush – force propagation of changed NIS map

SYNOPSIS

/usr/etc/yp/yppush [–**v**] [–**d** *domain*] *mapname*

DESCRIPTION

yppush copies a new version of a Network Interface Service (NIS) map from the master NIS server to the slave NIS servers. It is normally run only on the master NIS server by the **Makefile** in **/var/yp** after the master databases are changed. It first constructs a list of NIS server hosts by reading the NIS map **ypservers** within the *domain*. Keys within the map **ypservers** are the ASCII names of the machines on which the NIS servers run.

A "transfer map" request is sent to the NIS server at each host, along with the information needed by the transfer agent (the program which actually moves the map) to call back the **yppush .** When the attempt has completed (successfully or not), and the transfer agent has sent **yppush** a status message, the results may be printed to stdout. Messages are also printed when a transfer is not possible; for instance when the request message is undeliverable, or when the timeout period on responses has expired.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

OPTIONS

–**d** *domain*
: Specify a *domain*.

–**v**
: Verbose. This prints messages when each server is called, and for each response. If this flag is omitted, only error messages are printed.

FILES

/var/yp/*domain***/ypservers.**{*dir*,*pag*}
/var/yp

SEE ALSO

ypfiles(5), **ypserv**(8), **ypxfr**(8)

NIS protocol specification

BUGS

In the current implementation (version 2 NIS protocol), the transfer agent is **ypxfr**(8), which is started by the **ypserv** program. If **yppush** detects that it is speaking to a version 1 NIS protocol server, it uses the older protocol, sending a version 1 YPPROC_GET request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for version 1 servers. **yppush** prints a message saying that an "old-style" message has been sent. The system administrator should later check to see that the transfer has actually taken place.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypserv, ypbind, ypxfrd – NIS server and binder processes

SYNOPSIS

/usr/etc/ypserv [**–d**]

/usr/etc/ypbind [–s] [**–ypset** | **–ypsetme**]

ypxfrd [**–x**]

AVAILABILITY

This program is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

The Network Interface Service (NIS) provides a simple network lookup service consisting of databases and processes. The databases are **dbm**(3X) files in a directory tree rooted at **/var/yp**. These files are described in **ypfiles**(5). The processes are **/usr/etc/ypserv**, the NIS database lookup server, and **/usr/etc/ypbind**, the NIS binder. The programmatic interface to the NIS service is described in **ypclnt**(3N). Administrative tools are described in **yppush**(8), **ypxfr**(8), **yppoll**(8), **ypwhich**(8), and **ypset**(8). Tools to see the contents of NIS maps are described in **ypcat**(1), and **ypmatch**(1). Database generation and maintenance tools are described in **ypinit**(8), **ypmake**(8), and **makedbm**(8).

Both **ypserv** and **ypbind** are daemon processes typically activated at system startup time from **/etc/rc.local**. **ypserv** runs only on NIS server machines with a complete NIS database. **ypbind** runs on all machines using the NIS services, both NIS servers and clients.

ypxfrd transfers entire NIS maps in an efficient manner. For systems that use this daemon, map transfers will be 10 to 100 times faster, depending on the map. To use this daemon, **ypxfrd** should be run on a server running SunOS release 4.1. **ypxfr** will attempt to use **ypxfrd** first, if that fails, it will print a warning and then use the older transfer method.

The **ypserv** daemon's primary function is to look up information in its local database of NIS maps. The operations performed by **ypserv** are defined for the implementor by the *YP Protocol Specification*, and for the programmer by the header file **rpcsvc/yp_prot.h**. Communication to and from **ypserv** is by means of RPC calls. Lookup functions are described in **ypclnt**(3N), and are supplied as C-callable functions in the C library. There are four lookup functions, all of which are performed on a specified map within some NIS domain: **match, get_first, get_next**, and **get_all.** The **match** operation takes a key, and returns the associated value. The **get_first** operation returns the first key-value pair from the map, and **get_next** can be used to enumerate the remainder. **get_all** ships the entire map to the requester as the response to a single RPC request.

Two other functions supply information about the map, rather than map entries: **get_order_number**, and **get_master_name**. In fact, both order number and master name exist in the map as key-value pairs, but the server will not return either through the normal lookup functions. If you examine the map with **makedbm**(8), however, they will be visible. Other functions are used within the NIS service subsystem itself, and are not of general interest to NIS clients. They include **do_you_serve_this_domain?, transfer_map**, and **reinitialize_internal_state**.

The function of **ypbind** is to remember information that lets client processes on a single node communicate with some **ypserv** process. **ypbind** must run on every machine which has NIS client processes; **ypserv** may or may not be running on the same node, but must be running somewhere on the network.

The information **ypbind** remembers is called a *binding* — the association of a domain name with the internet address of the NIS server, and the port on that host at which the **ypserv** process is listening for service requests. This information is cached in the directory **/var/yp/binding** using a filename of **domainname.version**.

The process of binding is driven by client requests. As a request for an unbound domain comes in, the **ypbind** process broadcasts on the net trying to find a **ypserv** process that serves maps within that domain. Since the binding is established by broadcasting, there must be at least one **ypserv** process on every net. If the client is running in C2 secure mode, then **ypbind** will only accept bindings to servers where the **ypserv** process is running as root. Once a domain is bound by a particular **ypbind**, that same binding is given to every client process on the node. The **ypbind** process on the local node or a remote node may be queried for the binding of a particular domain by using the **ypwhich**(1) command.

Bindings and rebindings are handled transparently by the C library routines. If **ypbind** is unable to speak to the **ypserv** process it's bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind the domain once again. Requests received for an unbound domain will wait until the domain requested is bound. In general, a bound domain is marked as unbound when the node running **ypserv** crashes or gets overloaded. In such a case, **ypbind** will to bind any NIS server (typically one that is less-heavily loaded) available on the net.

ypbind also accepts requests to set its binding for a particular domain. The request is usually generated by the NIS subsystem itself. **ypset**(8) is a command to access the **set_domain** facility. It is for unsnarling messes. Note: the **set_domain** procedure only accepts requests from processes running as root.

OPTIONS

–d The NIS service should go to the DNS (Domain Name Service) for more host information.

–s Secure. When specified, only ypservers bound to a reserved port are used. This allows for a slight increase in security in completely controlled environments, where there are no computers operated by untrusted individuals. It offers no real increase in security.

–v Do not fork when **ypxfrd** is called multiple times.

–ypset **ypset**(8) may be used to change the binding. This option is very dangerous, and only should be used for debugging the network from a remote machine.

–ypsetme
ypset(8) may be issued from this machine, security is based on IP address checking, which can be defeated on network where untrusted individuals may inject packets. This option is not recommended.

FILES

If the file **/var/yp/ypserv.log** exists when **ypserv** starts up, log information will be written to this file when error conditions arise.

The file(s) **/var/yp/binding/domainname.version** will be created to speed up the binding process. These files cache the last successful binding created for the given domain, when a binding is requested these files are checked for validity and then used.
/var/yp
/usr/etc/ypbind

SEE ALSO

domainname(1), **ypcat**(1), **ypmatch**(1), **dbm**(3X), **ypclnt**(3N), **ypfiles**(5) **makedbm**(8), **ypmake**(8), **ypinit**(8), **yppoll**(8), **yppush**(8), **ypset**(8), **ypwhich**(8), **ypxfr**(8),

Network Programming
System and Network Administration

NOTES

Both **ypbind** and **ypserv** support multiple domains. The **ypserv** process determines the domains it serves by looking for directories of the same name in the directory **/var/yp**. It will reply to all broadcasts requesting yp service for that domain. Additionally, the **ypbind** process can maintain bindings to several domains and their servers, the default domain is however the one specified by the **domainname**(1) command at startup time.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypset – point ypbind at a particular server

SYNOPSIS

/usr/etc/yp/ypset [**–V1**|**–V2**] [**–d** *domain*] [**–h** *host*] *server*

DESCRIPTION

ypset tells **ypbind** to get the Network Interface Service (NIS) for the specified *domain* from the **ypserv** process running on *server*. If *server* is down, or is not running **ypserv**, this is not discovered until an NIS client process tries to get a binding for the domain. At this point, the binding set by **ypset** is tested by **ypbind**. If the binding is invalid, **ypbind** attempts to rebind for the same domain.

ypset is useful for binding a client node which is not on a broadcast net, or is on a broadcast net which is not running an NIS server host. It also is useful for debugging NIS client applications, for instance where an NIS map only exists at a single NIS server host.

In cases where several hosts on the local net are supplying NIS services, it is possible for **ypbind** to rebind to another host even while you attempt to find out if the **ypset** operation succeeded. For example, you can type:

```
example% ypset host1
example% ypwhich
host2
```

which can be confusing. This is a function of the NIS service subsystem's attempt to load-balance among the available NIS servers, and occurs when *host1* does not respond to **ypbind** because it is not running **ypserv** (or is overloaded), and *host2*, running **ypserv**, gets the binding.

server indicates the NIS server to bind to, and can be specified as a name or an IP address. If specified as a name, **ypset** attempts to use NIS services to resolve the name to an IP address. This works only if the node has a current valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

OPTIONS

–V1 Bind *server* for the (old) v.1 NIS protocol.

–V2 Bind *server* for the (current) v.2 NIS protocol.

If no version is supplied, **ypset**, first attempts to set the domain for the (current) v.2 protocol. If this attempt fails, **ypset**, then attempts to set the domain for the (old) v.1 protocol.

–h*host* Set ypbind's binding on *host*, instead of locally. *host* can be specified as a name or as an IP address.

–d*domain*
Use *domain* , instead of the default domain.

DIAGNOSTICS

Sorry, I couldn't send my rpc message to ypbind on host *name*
The user is not root, or **ypbind** was run without one of the **–ypset** flags. See **ypserv**(8) for explanations of the **–ypset** flags.

SEE ALSO

ypwhich(1), **ypfiles**(5), **ypserv**(8)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypsync – collect most up-to-date NIS maps

SYNOPSIS

/usr/etc/yp/ypsync [**–r**] [**–u**]

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

ypsync gathers current Network Information Service (NIS) maps to the local NIS server. When invoked with no arguments, it polls all the NIS servers listed in the **/etc/ypservers** NIS map for the maps they serve, and the order of those maps. If there are any new maps that the local server does not have, or if there are maps that are more current than the local server's copy, it excutes **ypxfr**(8) to transfer those maps to the local server.

ypsync eliminates the need for **cron**(8) jobs to ensure that NIS map updates are eventually transmitted to all NIS servers, and supports different NIS maps having different masters. It is invoked periodically by **ypserv**(8).

OPTIONS

–r When invoked with the **–r** flag, **ypsync** re-creates the local **/var/yp** directory and databases if needed. This facility is used when upgrading servers, since they can automatically retrieve NIS maps without needing manual intervention. The NIS master of the **ypservers** map can also designate new servers, which would automatically pick up their new maps on reboot.

–u When invoked with the **–u** flag, **ypsync** updates the list of NIS servers on the master of the **ypservers** NIS map to include the local system if it does not already, and then get copies of all the NIS databases. A user invoking **ypsync –u** may not be root, and must have the *networks privilege in the* NIS **group** map.

FILES

/var/yp/YP.*domainname*

SEE ALSO

ypupdate(3), **ypserv**(8), **ypxfr**(8)

Sun386i Advanced Administration

System and Network Administration

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypupdated, rpc.ypupdated – server for changing NIS information

SYNOPSIS

rpc.ypupdated [–is]

DESCRIPTION

ypupdated is a daemon that updates information in the Network Interface Service (NIS), normally started up by **inetd**(8C). **ypupdated** consults the file **updaters**(5) in the directory **/var/yp** to determine which NIS maps should be updated and how to change them.

By default, the daemon requires the most secure method of authentication available to it, either DES (secure) or UNIX (insecure).

OPTIONS

–i Accept RPC calls with the insecure AUTH_UNIX credentials. This allows programmatic updating of the NIS maps in all networks.

–s Accept only calls authenticated using the secure RPC mechanism (AUTH_DES authentication). This disables programmatic updating of the NIS maps unless the network supports these calls.

FILES

/var/yp/updaters

SEE ALSO

updaters(5), **inetd**(8C), **keyserv**(8C)

System and Network Administration
Network Programming

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

ypxfr – transfer NIS map from NIS server to here

SYNOPSIS

/usr/etc/yp/ypxfr [**–b**] [**–c**] [**–f**] [**–d** *domain*] [**–h** *host*] [**–s** *domain*] [**–C** *tid prog ipadd port*] *mapname*

DESCRIPTION

ypxfr moves a Network Interface Service (NIS) map in the default domain for the local host to the local host by making use of normal NIS services. It creates a temporary map in the directory **/var/yp/***domain* (this directory must already exist; *domain* is the default domain for the local host), fills it by enumerating the map's entries, fetches the map parameters (master and order number), and loads them. It then deletes any old versions of the map and moves the temporary map to the real *mapname*.

If run interactively, **ypxfr** writes its output to the terminal. However, if it is invoked without a controlling terminal, and if the log file **/var/yp/ypxfr.log** exists, it will append all its output to that file. Since **ypxfr** is most often run from the super-user's **crontab** file, or by **ypserv**, you can use the log file to retain a record of what was attempted, and what the results were.

If **issecure**(3) is TRUE, **ypxfr** requires that **ypserv** on the *host* be running as root. If the map being transferred is a secure map, **ypxfr** sets the permissions on the map to 0600.

For consistency between servers, **ypxfr** should be run periodically for every map in the NIS data base. Different maps change at different rates: the *services.byname* map may not change for months at a time, for instance, and may therefore be checked only once a day (in the wee hours). You may know that *mail.aliases* or *hosts.byname* changes several times per day. In such a case, you may want to check hourly for updates. A **crontab**(5) entry can be used to perform periodic updates automatically. Rather than having a separate **crontab** entry for each map, you can group commands to update several maps in a shell script. Examples (mnemonically named) are in **/usr/etc/yp**: **ypxfr_1perday**, **ypxfr_2perday**, and **ypxfr_1perhour**. They can serve as reasonable first cuts.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

OPTIONS

–b Preserve the resolver flag in the map during the transfer.

–c Do not send a "Clear current map" request to the local **ypserv** process. Use this flag if **ypserv** is not running locally at the time you are running **ypxfr**. Otherwise, **ypxfr** will complain that it cannot talk to the local **ypserv**, and the transfer will fail.

–f Force the transfer to occur even if the version at the master is not more recent than the local version.

–d *domain*
Specify a domain other than the default domain.

–h *host* Get the map from *host*, regardless of what the map says the master is. If *host* is not specified, **ypxfr** asks the NIS service for the name of the master, and tries to get the map from there. *host* may be a name or an internet address in the form *a.b.c.d*.

–s *domain*
Specify a source domain from which to transfer a map that should be the same across domains (such as the *services.byname* map).

–C*tid prog ipadd port*
This option is **only** for use by **ypserv**. When **ypserv** invokes **ypxfr**, it specifies that **ypxfr** should call back a **yppush** process at the host with IP address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

FILES

/var/yp/ypxfr.log log file

/usr/etc/yp/ypxfr_1perday
script to run one transfer per day, for use with **cron**(8)

/usr/etc/yp/ypxfr_2perday
script to run two transfers per day

/usr/etc/yp/ypxfr_1perhour
script for hourly transfers of volatile maps

/var/yp/*domain* NIS domain

/var/spool/cron/crontabs/root
Super-user's **crontab** file

SEE ALSO

issecure(3), **crontab**(5), **ypfiles**(5), **cron**(8), **ypserv**(8), **yppush**(8)

YP Protocol Specification, in *Network Programming*

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

zdump – time zone dumper

SYNOPSIS

zdump [–v] [**–c** *cutoffyear*] [*zonename* ...]

DESCRIPTION

zdump prints the current time in each *zonename* named on the command line.

OPTIONS

–v
: For each *zonename* on the command line, print the current time, the time at the lowest possible time value, the time one day after the lowest possible time value, the times both one second before and exactly at each time at which the rules for computing local time change, the time at the highest possible time value, and the time at one day less than the highest possible time value. Each line ends with **isdst=1** if the given time is Daylight Saving Time or **isdst=0** otherwise.

–c *cutoffyear*
: Cut off the verbose output near the start of the year *cutoffyear*.

FILES

/usr/share/lib/zoneinfo standard zone information directory

SEE ALSO

ctime(3V), **tzfile**(5), **zic**(8)

NAME

zic – time zone compiler

SYNOPSIS

zic [**–v**] [**–d** *directory*] [**–l** *localtime*] [*filename* ...]

DESCRIPTION

zic reads text from the file(s) named on the command line and creates the time conversion information files specified in this input. If a *filename* is '–', the standard input is read.

Input lines are made up of fields. Fields are separated from one another by any number of white space characters. Leading and trailing white space on input lines is ignored. An '#' (unquoted sharp character) in the input introduces a comment which extends to the end of the line the sharp character appears on. White space characters and sharp characters may be enclosed in ' " ' (double quotes) if they're to be used as part of a field. Any line that is blank (after comment stripping) is ignored. Non-blank lines are expected to be of one of three types: rule lines, zone lines, and link lines.

A rule line has the form

Rule NAME FROM TO TYPE IN ON AT SAVE LETTER/S

For example:

Rule USA 1969 1973 – Apr lastSun 2:00 1:00 D

The fields that make up a rule line are:

NAME Gives the (arbitrary) name of the set of rules this rule is part of.

FROM Gives the first year in which the rule applies. The word **minimum** (or an abbreviation) means the minimum year with a representable time value. The word **maximum** (or an abbreviation) means the maximum year with a representable time value.

TO Gives the final year in which the rule applies. In addition to **minimum** and **maximum** (as above), the word **only** (or an abbreviation) may be used to repeat the value of the **FROM** field.

TYPE Gives the type of year in which the rule applies. If **TYPE** is '–' then the rule applies in all years between **FROM** and **TO** inclusive; if **TYPE** is **uspres**, the rule applies in U.S. Presidential election years; if **TYPE** is **nonpres**, the rule applies in years other than U.S. Presidential election years. If **TYPE** is something else, then **zic** executes the command

yearistype *year type*

to check the type of a year: an exit status of zero is taken to mean that the year is of the given type; an exit status of one is taken to mean that the year is not of the given type.

IN Names the month in which the rule takes effect. Month names may be abbreviated.

ON Gives the day on which the rule takes effect. Recognized forms include:

5 the fifth of the month

lastSun the last Sunday in the month

lastMon the last Monday in the month

Sun>=8 first Sunday on or after the eighth

Sun<=25 last Sunday on or before the 25th

Names of days of the week may be abbreviated or spelled out in full. Note: there must be no spaces within the **ON** field.

AT Gives the time of day at which the rule takes effect. Recognized forms include:

2	time in hours
2:00	time in hours and minutes
15:00	24-hour format time (for times after noon)
1:28:14	time in hours, minutes, and seconds

Any of these forms may be followed by the letter **w** if the given time is local ''wall clock'' time or **s** if the given time is local ''standard'' time; in the absence of **w** or **s**, wall clock time is assumed.

SAVE Gives the amount of time to be added to local standard time when the rule is in effect. This field has the same format as the **AT** field (although, of course, the **w** and **s** suffixes are not used).

LETTER/S
Gives the ''variable part'' (for example, the ''S'' or ''D'' in ''EST'' or ''EDT'') of time zone abbreviations to be used when this rule is in effect. If this field is '–', the variable part is null.

A zone line has the form

Zone NAME GMTOFF RULES/SAVE FORMAT [UNTIL]

For example:

Zone Australia/South–west 9:30 Aus CST 1987 Mar 15 2:00

The fields that make up a zone line are:

NAME The name of the time zone. This is the name used in creating the time conversion information file for the zone.

GMTOFF
The amount of time to add to GMT to get standard time in this zone. This field has the same format as the **AT** and **SAVE** fields of rule lines; begin the field with a minus sign if time must be subtracted from GMT.

RULES/SAVE
The name of the rule(s) that apply in the time zone or, alternately, an amount of time to add to local standard time. If this field is '–' then standard time always applies in the time zone.

FORMAT
The format for time zone abbreviations in this time zone. The pair of characters **%s** is used to show where the ''variable part'' of the time zone abbreviation goes. **UNTIL** The time at which the GMT offset or the rule(s) change for a location. It is specified as a year, a month, a day, and a time of day. If this is specified, the time zone information is generated from the given GMT offset and rule change until the time specified.

The next line must be a ''continuation'' line; this has the same form as a zone line except that the string ''Zone'' and the name are omitted, as the continuation line will place information starting at the time specified as the **UNTIL** field in the previous line in the file used by the previous line. Continuation lines may contain an **UNTIL** field, just as zone lines do, indicating that the next line is a further continuation.

A link line has the form

Link LINK-FROM LINK-TO

For example:

Link **US/Eastern** **EST5EDT**

The **LINK-FROM** field should appear as the **NAME** field in some zone line; the **LINK-TO** field is used as an alternate name for that zone.

Except for continuation lines, lines may appear in any order in the input.

OPTIONS

−v
Complain if a year that appears in a data file is outside the range of years representable by system time values (0:00:00 AM GMT, January 1, 1970, to 3:14:07 AM GMT, January 19, 2038).

−d *directory*
Create time conversion information files in the directory **directory** rather than in the standard directory **/usr/share/lib/zoneinfo.**

−l *timezone*
Use the time zone *timezone* as local time. **zic** will act as if the file contained a link line of the form

Link *timezone* **localtime**

FILES

/usr/share/lib/zoneinfo standard directory used for created files

SEE ALSO

time(1V), **ctime**(3V), **tzfile**(5), **zdump**(8)

NOTES

For areas with more than two types of local time, you may need to use local standard time in the **AT** field of the earliest transition time's rule to ensure that the earliest transition time recorded in the compiled file is correct.

Index

9

A

B

E

H

M

T

U

V

Y