

CTIX™ OPERATING SYSTEM MANUAL

**Version C
Volume 3**

Convergent Technologies is a registered trademark of
Convergent Technologies, Inc.

Convergent, CTIX, S/80, S/280, S/480, S/640, and S/4040 are trademarks of
Convergent Technologies, Inc.

CTIX is derived from UNIX System V by Convergent Technologies under license from
AT&T. UNIX and RFS are trademarks of AT&T.

Material excerpted from the UNIX System V, Release 3.2 *System Administrator's/User's
Reference Manual* and *Programmer's Reference Manual* is Copyright 1989 by AT&T
Technologies. Reprinted by permission.

This software and documentation is based in part on the Fourth Berkeley Software
Distribution under license from the Regents of the University of California.

This manual was prepared on a Convergent Technologies S/640 Computer System and
was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02264-01
Update Notice 1 (November 1990) 09-02578

Copyright © 1990 by Convergent Technologies, Inc.,
San Jose, CA. Printed in USA.

All rights reserved. No part of this document may be reproduced, transmitted, stored in a
retrieval system, or translated into any language without the prior written consent of
Convergent Technologies, Inc.

Convergent Technologies makes no representations or warranties with respect to the
contents hereof and specifically disclaims any implied warranties of merchantability or
fitness for any particular purpose. Further, Convergent Technologies reserves the right
to revise this publication and to make changes from time to time in its content without
being obligated to notify any person of such revision or changes.

TABLE OF CONTENTS: VOLUME 3

How to Use This Manual ix

2. System Calls

intro	introduction to system calls and error numbers
access	determine accessibility of a file
acct	enable or disable process accounting
adjtime	correct the time to allow synchronization of the system clock
alarm	set a process alarm clock
bind	bind a name to a socket
brk	change data segment space allocation
chdir	change working directory
chmod	change mode of file
chown	change owner and group of a file
chroot	change root directory
close	close a file descriptor
connect	initiate a connection on a socket
creat	create a new file or rewrite an existing one
dup	duplicate an open file descriptor
exec	execute a file
exit	terminate process
fcntl	file control
fork	create a new process
getdents	read directory entries and put in a file
getdtablesize	get descriptor table size
gethostid	get/set unique identifier of current host
gethostname	get/set name of current host
getmsg	get next message off a stream
getpeername	get name of connected peer
getpid	get process, process group, and parent process IDs
getsockname	get socket name
getsockopt	get and set options on sockets
gettimeofday	get/set date and time
getuid	get real user, effective user, real group, and effective group IDs
ioctl	control device
kill	send a signal to a process or a group of processes
link	link to a file
listen	listen for connections on a socket
locking	exclusive access to regions of a file
lseek	move read/write file pointer
mkdir	make a directory
mknod	make a directory, or a special or ordinary file
mount	mount a file system
msgctl	message control operations
msgget	get message queue
msgop	message operations
nfssys	common shared NFS system calls
nice	change priority of a process

notify manage notifications
 open open for reading or writing
 pause suspend process until signal
 pipe create an interprocess channel
 plock lock process, text, or data in memory
 poll STREAMS input/output multiplexing
 profil execution time profile
 ptrace process trace
 putmsg send a message on a stream
 read read from file
 recv receive a message from a socket
 rmdir remove a directory
 select synchronous I/O multiplexing
 semctl semaphore control operations
 semget get set of semaphores
 semop semaphore operations
 send send a message to a socket
 setpgrp set process group ID
 setuid set user and group IDs
 shmctl shared memory control operations
 shmget get shared memory segment identifier
 shmop shared memory operations
 shutdown shut down part of a full-duplex connection
 signal specify what to do upon receipt of a signal
 sigset signal management
 socket create an endpoint for communication
 stat get file status
 statfs get file system information
 stime set time
 swrite synchronous write on a file
 sync update super block
 sysfs get file system type information
 syslocal special system requests
 time get time
 times get process and child process times
 uadmin administrative control
 ulimit get and set user limits
 umask set and get file creation mask
 umount unmount a file system
 uname get name of current CTIX system
 unlink remove directory entry
 ustat get file system statistics
 utime set file access and modification times
 wait wait for child process to stop or terminate
 write write on a file

3. Subroutines and Libraries

intro introduction to functions and libraries
 a64l convert between long integer and base-64 ASCII string
 abort generate a SIGABRT
 abs return integer absolute value

assert verify program assertion
 bessel Bessel functions
 bsearch binary search a sorted table
 bstring bit and byte string operations
 byteorder convert values between host and network byte order
 clock report CPU time used
 conv translate characters
 crypt generate hashing encryption
 crypt password and file encryption functions
 ctermid generate file name for terminal
 ctime convert date and time to string
 ctype character handling
 curses terminal screen handling and optimization package
 cuserid get character login name of the user
 dbm database subroutines
 dial establish an out-going terminal line connection
 directory directory operations
 drand48 generate uniformly distributed pseudo-random numbers
 dup2 duplicate an open file descriptor
 ecvt convert floating-point number to string
 end last locations in program
 erf error function and complementary error function
 exp exponential, logarithm, power, square root functions
 fclose close or flush a stream
 ferror stream status inquiries
 floor floor, ceiling, remainder, absolute value functions
 fopen open a stream
 fpgetround IEEE floating point environment control
 fread binary input/output
 frexp manipulate parts of floating-point numbers
 fseek reposition a file pointer in a stream
 ftw walk a file tree
 gamma log gamma function
 getc get character or word from a stream
 getcwd get path-name of current working directory
 getenv return value for environment name
 getgrent get group file entry
 gethostbyname get network host entry
 getlogin get login name
 getnetent get network entry
 getopt get option letter from argument vector
 getpass read a password
 getprotoent get protocol entry
 getpw get name from UID
 getpwent get password file entry
 getrpcent get rpc entry
 getrpcport get RPC port number
 gets get a string from a stream
 getservent get service entry
 getspent get shadow
 getut access utmp file entry
 hsearch manage hash search tables

hypot Euclidean distance function
 inet Internet address manipulation routines
 isnan test for floating point NaN (Not-A-Number)
 l3tol convert between 3-byte integers and long integers
 ldahread read the archive header of a member of an archive file
 ldclose close a common object file
 ldhread read the file header of a common object file
 ldgetname retrieve symbol name for common object file symbol table entry
 ldread manipulate line number entries of a common object file function
 ldseek seek to line number entries of a section of a common object file
 ldohseek seek to the optional file header of a common object file
 ldopen open a common object file for reading
 ldrseek seek to relocation entries of a section of a common object file
 ldshread read an indexed/named section header of a common object file
 ldsseek seek to an indexed/named section of a common object file
 ldtbindex compute the index of a symbol table entry of a common object file
 ldtbread read an indexed symbol table entry of a common object file
 ldtbseek seek to the symbol table of a common object file
 libdev manipulate Volume Home Blocks (VHB)
 lockf record locking on files
 logname return login name of user
 lsearch linear search and update
 malloc main memory allocator
 malloc fast main memory allocator
 matherr error-handling function
 memory memory operations
 mktemp make a unique file name
 monitor prepare execution profile
 ndbm database subroutines
 nlist get entries from name list
 nlsgetcall get client's data passed through the listener
 nlsprovider get name of transport provider
 nlsrequest format and send listener service request message
 ocurse optimized screen functions
 otermcap terminal independent operations
 perror system error messages
 plot graphics interface subroutines
 popen initiate pipe to/from a process
 printf print formatted output
 putc put character or word on a stream
 putenv change or add value to environment
 putpwent write password file entry
 puts put a string on a stream
 putspent write shadow password file entry
 qsort quicker sort
 rand simple random-number generator
 rcmd routines for returning a stream to a remote command
 regcmp compile and execute regular expression
 resolver resolver routines
 rexec return stream to a remote command
 scanf convert formatted input
 setbuf assign buffering to a stream

setjmp non-local goto
 sinh hyperbolic functions
 sleep suspend execution for interval
 sputl access long integer data in a machine-independent fashion
 ssignal software signals
 stdio standard buffered input/output package
 stdipc standard interprocess communication package
 string string operations
 strtod convert string to double-precision number
 strtol convert string to integer
 swab swap bytes
 system issue a shell command
 t_accept accept a connect request
 t_alloc allocate a library structure
 t_bind bind an address to a transport endpoint
 t_close close a transport endpoint
 t_connect establish a connection with another transport user
 t_error produce error message
 t_free free a library structure
 t_getinfo get protocol-specific service information
 t_getstate get the current state
 t_listen listen for a connect request
 t_look look at the current event on a transport endpoint
 t_open establish a transport endpoint
 t_optmgmt manage options for a transport endpoint
 t_rcv receive data or expedited data sent over a connection
 t_rcvconnect receive the confirmation from a connect request
 t_rcvdis retrieve information from disconnect
 t_rcvrel acknowledge receipt of an orderly release indication
 t_rcvudata receive a data unit
 t_rcvuderr receive a unit data error indication
 t_snd send data or expedited data over a connection
 t_snddis send user-initiated disconnect request
 t_sndrel initiate an orderly release
 t_sndudata send a data unit
 t_sync synchronize transport library
 t_unbind disable a transport endpoint
 tmpfile create a temporary file
 tmpnam create a name for a temporary file
 trig trigonometric functions
 tsearch manage binary search trees
 ttyname find name of a terminal
 ttyslot find the slot in the utmp file of the current user
 ungetc push character back into input stream
 vprintf print formatted output of a varargs argument list

—

—

—

NAME

fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky - IEEE floating-point environment control

SYNOPSIS

```
#include <ieeefp.h>

typedef enum {
    FP_RN=0,      /* round to nearest */
    FP_RZ=0x10,   /* round to zero (truncate) */
    FP_RM=0x20,   /* round to minus */
    FP_RP=0x30,   /* round to plus */
} fp_rnd;

fp_rnd fpgetround();

fp_rnd fpsetround(rnd_dir)
fp_rnd rnd_dir;

#define    fp_except    int
#define    FP_X_INV     0x80 /* invalid operation */
                        /* exception */
#define    FP_X_OFL     0x40 /* overflow */
                        /* exception */
#define    FP_X_UFL     0x20 /* underflow */
                        /* exception */
#define    FP_X_DZ      0x10 /* divide-by-zero */
                        /* exception */
#define    FP_X_IMP     0x08 /* imprecise (loss */
                        /* of precision) */

fp_except fpgetmask();

fp_except fpsetmask(mask);
fp_except mask;

fp_except fpgetsticky();

fp_except fpsetsticky(sticky);
fp_except sticky;
```

DESCRIPTION

These routines let the user change the behavior on the occurrence of any of five floating-point exceptions: divide-by-zero, overflow, underflow, imprecise (inexact) result, and invalid operation. The routines also change the rounding mode for floating-point operations. When a floating-point exception occurs,

the corresponding sticky bit is set (1), and if the mask bit is enabled (1), the trap takes place. The routines are valid only on systems that are equipped with floating-point accelerator hardware; otherwise, floating-point operations are compiled differently and handled in software.

The *fpgetround()* routine returns the current rounding mode.

The *fpsetround()* routine sets the rounding mode and returns the previous rounding mode.

The *fpgetmask()* routine returns the current exception masks.

The *fpsetmask()* routine sets the exception masks and returns the previous setting.

The *fpgetsticky()* routine returns the current exception sticky flags.

The *fpsetsticky()* routine sets (clears) the exception sticky flags and returns the previous setting.

The environment for Convergent computers with either a MC68040 CPU or a combined MC68020 CPU with MC68881 or MC68882 floating-point processor follows:

- Rounding mode set to nearest(FP_RN)
- Divide-by-zero
- Floating-point overflow
- Invalid operation traps enabled

SEE ALSO

isnan(3C).

CAVEATS

The utilities described in this man page are applicable only for computers that are equipped with either the MC68040 microprocessor, or both the MC68020 microprocessor CPU and the MC68881, or the MC68882 microprocessor for a hardware-floating point accelerator. Programs that invoke these utilities are run on computers without the floating-point hardware and result in no operation and no returned error message for the particular function.

One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.

For the same reason, when calling *fpsetmask()*, the user should make sure that the sticky bit corresponding to the exception being enabled is cleared.

WARNINGS

The *fpsetsticky()* routine modifies all sticky flags; *fpsetmask()* changes all mask bits.

C requires truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.

—

—

—

NAME

getspent, getsnam, setspent, endspent, fgetspent, lckpwn, ulckpwn - get shadow password file entry

SYNOPSIS

```
#include <shadow.h>

struct spwd *getspent ()

struct spwd *getsnam (name)
char *name;

int lckpwn ()
int ulckpwn ()

void setspent ()

void endspent ()

struct spwd *fgetspent (fp)
FILE *fp;
```

DESCRIPTION

The *getspent* and *getsnam* routines each return a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/shadow* file. Each line in the file contains a shadow password structure (*spwd*), declared in the *<shadow.h>* header file:

```
struct spwd{
    char    *sp_namp;
    char    *sp_pwdp;
    long    sp_lstchg;
    long    sp_min;
    long    sp_max;
};
```

The *getspent* routine, when first called, returns a pointer to the first *spwd* structure in the file; thereafter, it returns a pointer to the next *spwd* structure in the file. This way, successive calls can be used to search the entire file. The *getsnam* routine searches from the beginning of the file until a login matching *name* is found, and then returns a pointer to the particular structure in which it was found. The *getspent* and *getsnam* routines populate the *sp_min* or *sp_max* field with -1 if the corresponding field in */etc/shadow* is empty. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

The */etc/.pwd.lock* file is the lock file, which is used to coordinate modification access to the password files in */etc/passwd* and */etc/shadow*. The *lckpwn()*

and *ulckpwordf()* routines are used to gain modification access to the password files, through the lock file. A process first uses *lckpwordf()* to lock the lock file, thereby gaining exclusive rights to modify the */etc/passwd* or */etc/shadow* file. Upon completing modifications, a process should release the lock on the lock file by using *ulckpwordf()*. This lock mechanism prevents simultaneous modification of the password files.

The *lckpwordf()* routine attempts to lock the file */etc/.pwd.lock*. If the file is already locked, *lckpwordf()* tries for 15 seconds to lock the file. If unsuccessful, *lckpwordf()* returns a -1; if successful within 15 seconds, *lckpwordf()* returns a return code other than -1.

The *ulckpwordf()* routine attempts to unlock the file */etc/.pwd.lock*. If successful, *ulckpwordf()* returns a 0; if unsuccessful (if the file is not locked), *ulckpwordf()* returns a -1.

A call to the *setspent* routine has the effect of rewinding the shadow password file to allow repeated searches. The *endspent* routine may be called to close the shadow password file when processing is complete.

The *fgetspent* routine returns a pointer to the next *spwd* structure in the stream *fp*, which matches the format of */etc/shadow*.

FILES

/etc/shadow
/etc/passwd
/etc/.pwd.lock

SEE ALSO

putspent(3X).

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

CAVEAT

All information is contained in a static area, so it must be copied if it is to be saved.

WARNING

If a program not otherwise using standard I/O uses this routine, the size of the program increases more than might be expected.

This routine is for internal use only; compatibility is not guaranteed.

NAME

monitor - prepare execution profile

SYNOPSIS

```
#include <mon.h>
```

```
void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
WORD *buffer;
int bufsize, nfunc;
```

DESCRIPTION

An executable program created by `cc-p`, it automatically includes calls for `monitor` with default parameters; `monitor` need not be called explicitly.

`monitor` is an interface to `profil(2)`. `lowpc` and `highpc` are the addresses of two functions; `buffer` is the address of a (user-supplied) array of `bufsize` WORDs (defined in the `<mon.h>` header file). `monitor` arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of `lowpc` and the highest is just below `highpc`. `lowpc` may not equal 0 for this use of `monitor`. At most, `nfunc` call counts can be kept; only calls of functions compiled with the profiling option `-p` of `cc(1)` are recorded.

`prof(1)` can then be used to examine the results.

The name of the file written by `monitor` is controlled by the environment variable `PROFDIR`. If `PROFDIR` does not exist, `mon.out` is created in the current directory. If `PROFDIR` exists but has no value, `monitor` does not do any profiling and creates no output file. Otherwise, the value of `PROFDIR` is used as the name of the directory in which to create the output file. If `PROFDIR` is `dirname`, then the file written is `dirname/pid.mon.out`, where `pid` is the program's process ID. (When `monitor` is called automatically by compiling via `cc -p`, the file created is `dirname/pid.progname`, where `progname` is the name of the program.)

The following discussion is a sketch of `monitor` usage.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no less than one half of the range of locations sampled.

To profile the entire program, put the following at the start of `main()` :

```
extern etext;
...
monitor ((int (*)())2, &etext, buf, bufsize, nfunc);
```

etext lies just above all the program text; see *end(3C)*.

To stop execution monitoring and write the results, put the following at the end of `main()` :

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

Do not compile with the `-p` option. Run the program and use *prof(1)* to view the results in the output file `mon.out` .

FILES

`mon.out`

SEE ALSO

`cc(1)`, `prof(1)`, `profil(2)`, `end(3C)`.

BUGS

The “*dirname/pid.mon.out*” form does not work; the “*dirname/pid.progname*” form (automatically called via `cc -p`) does work.

NAME

sleep - suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) because scheduled wakeups occur at fixed 1-second intervals (on the second, according to an internal clock), and (2) because any caught signal terminates the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling 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.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the *sleep* routine returns. But if the *sleep* time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

SEE ALSO

alarm(2), pause(2), signal(2).

WARNING

sleep uses *signal(2)*, not *sigset(2)*, to reset the caller's **SIGALRM** handler routine. Therefore, the signal action is reset to its default action on execution of the **SIGALRM** handler. This is probably not what the programmer intended if *sigset(2)* had originally been used to set the signal action.

sleep uses a longjmp, which returns to the *sleep* context when the *alarm(2)* signal handler routine is executed. This may cause premature preemption and loss of context from other nested signal handler routines.

—

—

—