REGULUS

PROGRAMMER'S REFERENCE MANUAL

Alcyon CORPORATION

REGULUS™
PROGRAMMER'S REFERENCE MANUAL

Version 4.1

October 1983

Alcyon Corporation
8716 Production Avenue
San Diego, California 92121

INTRODUCTION

REGULUS is a UNIX-like operating system. REGULUS supports several important features which are not standard UNIX features such as real-time tasks, shared memory data segments, user task access to physical memory segments, etc. A careful reading of this manual will explain these and other REGULUS features. Some of the lesser used UNIX utilities are not provided with REGULUS. See Section I (CMND) for a complete list of REGULUS utilities. See also the REGULUS C compiler documentation (CC(CMND)) for implementation details which may differ from other C compilers.

GETTING STARTED WITH REGULUS
ALCYON APS Version

I. Bringing REGULUS up initially

REGULUS for the Alcyon APS system is delivered on a bootable hard disk and also on a bootable hard disk cartridge. To bring up REGULUS on your system, choose the boot command from the following list depending on which hard disk is on your system:

Command	Disk
BO	5MB DMA Fixed
G FE2200	5MB DMA Fixed
G FE2000	5MB DMA Cartridge
G FE2400	80MB Fujitsu Fixed
G FE2800	33MB Atasi Fixed
G FE3000	5MB Syquest Drive 0
G FE3200	5MB Syquest Drive 1
G FE3400	46MB Atasi Fixed

This command will read a copy of REGULUS from the disk into memory and execute it. REGULUS will type the following lines on the console terminal (your serial number on the first line (ie. #5932) will be different):

```
Regulus 4.1 8/12/83 #5932
Copyright Alcyon Corp.
@(#) ALC Version of Tue Oct 25 17:20 1983
sys/non/wst 128K/384K/856
#
```

The first two lines identify the system, and the third line identifies the date and time when this version of REGULUS was generated. The fourth line gives memory sizes: in this example, 96K is the size of the operating system, 416K is the size of the memory available for user programs, and 16 bytes are yet available in this version of REGULUS before it crosses the next 4K byte boundary.

REGULUS is now executing in "single user" mode. This means that only the console terminal is enabled, and the super-user "root" is automatically logged on. To bring the system up multi-user, you type "exit" on the console. As delivered, REGULUS is only configured for one user on the console terminal, and only the user "root" with no password. To add additional users and terminals, see the 'users (misc)' document.

II. Copying REGULUS file systems

The DMA hard disk is partitioned into two logical disks under

REGULUS: /dev/dmaf is the name of the fixed portion and /dev/dmar is the name of the removable portion. In order to copy the REGULUS file system from the removable cartridge to the fixed portion of the disk, execute the following commands while running in single user mode:

```
cd /install
carttofix
sync
```

The "carttofix" command copies the REGULUS file system from the removable portion of the disk to the fixed portion and makes the fixed disk bootable.

Typically the fixed disk of the APS contains an exact copy of the distribution cartridge disk, so the above command is unnecessary.

The DMA fixed disk now contains a bootable copy of REGULUS and its file system. This fixed disk can now be used as the root file system by booting directly from it by typing the following command to the MACSbug monitor:

```
BO
```

If you have a running REGULUS file system on the fixed portion of the DMA disk and wish to build a bootable copy of this file system on a cartridge, use the following set of commands:

```
cd /install
fixtocart
sync
```

If your system is configured with only the DMA disk, these two command files are the recommended backup/restore procedure because they not only make a copy of all files but also make the copy bootable.

If you have the larger fixed disk (Atasi 33MB or 46MB), you should use the "tar" command to backup files from the hard disk to the cartridge (see tar(cmnd) document) ie.

```
cd / tar -cvsf 10560 /dev/rdmar .
```

To restore such a tar dump, also use the command tar(cmnd) ie.

```
cd / tar -xvsf 10560 /dev/rdmar
```

The special file names for each of the available disks are:

File Name	Disk
/dev/dmar	5MB DMA Cartridge
/dev/dmaf	5MB Dma Fixed

```
/dev/as20       33MB or 46MB Atasi Fixed
/dev/syq0       5MB Syquest Cartridge Drive 0
/dev/syq1        5MB Syquest Cartridge Drive 1
```

III. Configuring REGULUS for your installation

As distributed on the disk cartridge, the REGULUS file system is configured to use 8960 disk blocks of 512 bytes each. If your fixed disk contains more than 8960 sectors of 512 bytes each, you can configure the REGULUS file system on the fixed disk to use more space with the following command sequence:

```
setfssize /dev/as20 40000
fsck -s /dev/as20
```

This will change the file system size to be 40000 blocks of 512 bytes each and will rebuild the free block list on that file system. If you are executing with /dev/ata2 as the REGULUS root file system when these commands are executed, reboot the system after the fsck without doing a sync.

If you change the size of the fixed file system, you must also change the parameter fswapblock in the version of REGULUS which uses the fixed disk as its root file system so that no overlap occurs between the file system and the swapping area using a command sequence such as:

```
ddt /regulus
0!fswapblock
40000!fswapblock+2
;f
```

For more information about configuring disk drives, see the 'dma (dev)' or 'atasi (dev)' documents. For more information about configuring serial ports, see the 'tty (dev)' and 'duart (dev)' documents. For more information about configuring REGULUS run-time paramters, see 'config (misc)'.

IV. Backing up REGULUS file systems

It is important that you keep a bootable REGULUS cartridge as backup in case the copy on your active disk gets destroyed. The cartridge on which REGULUS was delivered will serve this function well. There are two methods of backing up your active file system. If this file system will fit entirely on one removable cartridge, the command sequence given above for copying from fixed to removable may be used to produce a full backup of all files. The cartridge produced by this sequence will also serve as the bootable backup. The shell command "tar" may be used to create full or partial dumps of active file systems and to restore them. This program (tar) only functions with a running REGULUS system, thus the need for a bootable backup. To rebuild a REGULUS file system on the fixed portion of the disk from a full dump, boot from the cartridge backup, use "mkfs" to format an empty file system on "/dev/as20" (fixed

disk), and then use "tar" to restore the files from a backup dump. "tar" can use the hard disk cartridge for an input or output device.

Some other device names of interest are: /dev/tty? is the name of a terminal device (/dev/tty0 is the console; /dev/tty1, /dev/tty2, and /dev/tty3 are the other serial ports on the cpu board). /dev/lp is the printer.

At this point, a careful reading of the REGULUS Programmers Manual is advised if the reader is not familiar with UNIX or UNIX-like systems.

(sys) sbrk - add 'n' bytes to task break
(v6/sys) profil - profile task execution
(v7/sys) profil - profile task execution
(sys) ptrace - trace child task for debugging
(sys) nice - change task priority
(v6/sys) nice - change task priority
(v7/sys) nice - change task priority
(sys) getpid - get current task process id
(sys) ssignal - send signal to task (REGULUS)
(sys) trace - trace child task (REGULUS)
(sys) wait - wait for child task to terminate
(cmnd) env - execute task with specified environment
(sys) exit - end task with specified status
(maint) update - disk I/O integrity task
(subs) twait - wait for ended or stopped task
(sys) chdir - change working directory of task
(sys) fork - create copy of a running task
(sys) getegid - get effective group id of task
(sys) geteuid - get effective user id of task
(sys) getgid - get group id of task
(sys) getpgrp - get process group id of task
(sys) getuid - get user id of task
(sys) maketask - make a new task
(sys) setgid - set group id of task
(sys) setpgrp - set the process group of a task
(sys) setuid - set user id of task
(sys) times - return execution times of task
(v6/sys) times - return execution times of task
(v7/sys) times - return execution times of task
(cmnd) tee - enable multi-directional output
(subs) mktemp - make a temporary file name
(cmnd) stty - set terminal characteristics
(sys) gtty - get terminal control information
(sys) stty - set terminal control information
(v6/sys) gtty - get terminal control information
(v6/sys) stty - set terminal control information
(v7/sys) gtty - get terminal control information
(v7/sys) stty - set terminal control information
(cmnd) tty - display terminal name
(stdio) pclose - terminate i/o to/from a pipe
(sys) wait - wait for child task to terminate
(cmnd) ed - line oriented text editor
(cmnd) roff - text formatter
(cmnd) unrot - unrotate text lines
(sys) time - get time of day
(cmnd) time - time a command
(cmnd) time - time a command
(subs) ctime - convert time and date to ascii
(sys) ftime - get time of day (BSD compatible)
(subs) gettime - get time of day
(subs) tod - time of day
(sys) stime - set time of day
(sys) time - get time of day
(cmnd) date - print the current date and time
(cmnd) more - print files a page at a time

COMMANDS

This section contains most of the commands used in REGULUS. They are listed alphabetically.

Command entries are described in this section in the following sequence:

PROGRAM	Repeats the name of the entry and briefly states it's purpose.
USAGE	Summarizes the use of the program.
FUNCTION	Describes the program in detail.
FILES	Lists the file names that are built into the program.
DIAGNOSTICS	Describes routine for diagnosing programming errors.
SEE ALSO	Gives pointers to related information.
BUGS	Lists known program defects or errors.

PROGRAM
ar - archive maintainer

USAGE
ar flags [pos] archive file1 file2 ...

FUNCTION
'ar' collects individual files into a single archive, or library file. It is primarily used to maintain libraries of object files for use by the loader, although it can be used with any type of file.

The flags argument determines the function that 'ar' performs. One character from the set {dprtx} must be specified; these are the function select flags. The function select flags specify the following functions:

d Delete the specified files from the archive.

p Print the specified files. If the files are not printable, e.g. object files, havoc results.

r Replace the specified files. If an optional position flag from the set {abi} is not specified new files are placed at the end of the archive.

t Print a table of contents of the archive. If files are specified only the specified files are listed.

x Extract the specified files from the archive, creating loose files in the current directory. If no files are specified, all of the files in the archive are extracted. Note that 'x' does not alter the archive. The 'd' flag must be used to delete files from an archive.

The files must be specified in the same order as they appear in the archive file, otherwise some of the files may not be found. For example, if an archive "t.a" consists of the files file1, file2 and file3 in that order, the command "ar t t.a file2 file1" will cause file2 to be listed, but not file1. Instead, the message "file1 not in archive" will be printed. This is because 'ar' only searches the archive once. Files must be specified in the order they appear in the archive.

Characters from the set {abiv} may be included in the flags argument. These characters have the following function:

a When used with the 'r' flag, 'a' specifies that new files are to be inserted after the file "pos" in

the archive. Note that the "pos" argument must be specified if and only if one of the characters {abi} appear in the flags argument.

b 'b' has the same function as 'a', except that new files are inserted before the file "pos" in the archive.

i 'i' has exactly the same meaning as 'b', the two characters can be used interchangeably (but only one must appear).

v 'v' (verbose) causes a file by file description of the operation of 'ar' to be printed. Files are either copied (c), replaced (r), added (a), deleted (d) or extracted (x) from an archive. When used with the 't' flag, 'v' causes a long listing to be printed. Verbose has no effect on the print function.

FILES

_~ar????? (process id in octal) - temporary file

DIAGNOSTICS

SEE ALSO

ld (cmnd), ar (files)

BUGS

When using any of the 'a' 'b', or 'i' flags to position the files, if any of the files in the namelist already exist in the archive, you must delete them with the delete option first. Otherwise, 'ar' will make another copy of them.

PROGRAM

as - assembler for the Motorola 68000

USAGE

as [-i] [-p] [-u] [-L] [-N] [-T] sourcefile

FUNCTION

'as' is the REGULUS 68000 assembler. It is used to assemble a program for the Motorola 68000. The syntax accepted is identical to the Motorola 68000 cross assembler as described in Motorola Manual M68KXASM(D3) with the exceptions and additions described below. The source file must be present as the first argument. The assembler always produces a relocatable object file whose name is the same as the source file primary name with a ".o" extension -- ie. if the source file name is "test.s" then the object file name is "test.o"; if the source file name is "pgm1" then the object file name is "pgm1.o".

The -i option is used to initialize the assembler in the case that the file '/lib/as68symb' does not exist or becomes corrupted. It also requires the use of the initialization file /lib/as68init. The command would be of the form : 'as -i /lib/as68init'.

If the -p flag is specified, the assembler produces a hexidecimal side-by-side listing on the standard output (you may want to redirect this to a file). Error messages are also produced on the standard output file whether or not the -p flag is specified.

If the -u flag is specified, all undefined symbols in the assembly are treated as global.

If the -L flag is specified, all address constants are generated as 32-bit numbers. Default is 16-bit numbers where possible.

If the -N flag is specified, pass 1.5 of the assembler is not executed. This pass changes all long relative branches to short relative branches where possible.

If the -T flag is specified the assembler generates code suitable for the 68010. Let's the assembler know the proper meaning of the "movec", "moves", rte", and "rtd" instructions.

Error messages begin with an &, indicate the source line on which the error occurred and are meant to be self-explanatory. The error messages preceded by a single '&' are generated during the first pass of the assembler and those preceded by a pair '&&' are generated during the second pass of the assembler.

The assembler accepts both upper and lower case characters.

Labels and variables are case sensitive ('LOOP' is different from 'Loop'), but mnemonics and directives can be in either upper or lower case.

Macros are implemented using the C preprocessor which is part of the cc command (see CC(cmnd)). Conditional assembly is also implemented using the preprocessor. Use the command "cc -P pgm.s" to expand macros using the C preprocessor. The expanded file is left on pgm.i.

FILES

/lib/as68symb
/tmp/a6????A (???? is the process id number)
/lib/as68init

SEE ALSO

as (misc), lo (cmnd), cc(cmnd)

EXTRAS

The following enhancements have been added to aid the assembly language programmer by making the assembly language more regular:

move, add, sub mnemonics will actually generate moveq, addq, and suba instructions where possible. If a move instruction rather than a moveq instruction is desired (affecting only lower byte or word of D register), the size attribute must be explicitly coded ie. move.b or move.w. The assembler will change any move or move.l to moveq if possible.

clr.x An is allowed and will actually generate a suba.x An,An instruction.

add, sub, cmp with an A register source/destination are allowed and generate adda, suba, cmpa.

add, and, cmp, eor, or, sub are allowed with immediate first operands and actually generate addi, andi, cmpi, eori, ori, subi instructions if the second operand is not register direct.

All branch instructions generate short relative branches where possible, including forward references.

Any shift instruction with no shift count specified assumes a shift count of one; ie. "asl rl" is equivalent to "asl #1,rl".

jsr instructions are changed to bsr instructions if the resulting bsr is shorter than the jsr.

The mnemonics 'inc' and 'dec' are also recognized and are functionally equivilent to "addq 1" or "subq 1".

Several additional mnemonics have been added to the condition code instructions which map to the standard set (bt -> bra, bhs -> bhis, bnz -> bne, bze -> beq, dbhs -> dbhi, dblo -> dbcs, dbnz -> dbne, dbze -> dbeq, shs -> scc, slo -> scc, snz -> sne, sze -> seq).

BUGS

.set is currently implemented as a .equ.

There are several directives which are recognized but ignored: mask2, idnt, ttl, opt, and page.

PC relative addressing mode is currently not supported.

PROGRAM

as68 - assembler for the Motorola 68000

USAGE

as68 [-i] [-p] [-u] [-L] [-N] sourcefile

FUNCTION

'as68' is the REGULUS 68000 assembler. It is identical to the 'as' command and should be linked (see ln (cmnd)) to '/bin/as' under REGULUS systems.

For more information on this command, see the documentation under 'as' (cmnd).

SEE ALSO

as (cmnd)
c68 (cmnd) which is a link to '/bin/cc'
lo68 (cmnd) which is a link to '/bin/ld'

PROGRAM

build - process a file according to predefined specifications

USAGE

```
build [-b build-file] [-s] [-n] [keyword...]
build [-f build-file] [-s] [-n] [keyword...]
```

FUNCTION

Generally used to compile files according to a pre-specified command line. The "keyword" is actually a known keyword which resides in the specification line. "build" looks in the "build-file" for a line which has the specified keyword in it, and generates that line as an exec. If the "-b" option is not specified "build" looks for a file called 'buildfile' or 'Buildfile' in the current directory. The "-f" flag may be used instead of the "-b" option with the same results (for the build-file specifications). A build file entry might look like one of the following:

```
cc build.c -l7 -o build
cc -c -L atoi.c ; ar -rv lib7.a atoi.o
```

"build" echo's the executed command to the screen prior to the actual execution. Buildfile commands may contain multiple commands separated by semi-colons. If the file name is omitted "build" will treat the build file like a command file and will echo each command line to the screen before it is executed. More than one keyword can be specified in a single "build" command.

The '-s' flag causes "build" to execute only the first case of a 'keyword' in the file. Otherwise "build" looks for all instances of the 'keyword' in the build file. This is useful for building of multi-file programs, for example:

```
mkver -e "tester - "
cc -L -r main.c foo.c -l6 -l7 -o tester
setstack tester 8192 8192
```

The '-n' flag inhibits the actual execution of the command, only "build's" intentions are listed on the standard output device. The pound sign '#' as the first character on a line causes the line to be considered a comment. This line will be echoed, but will not be exec'd.

FILES

buildfile - default specification file
/usr/regulus/Buildfile - default specification file

SEE ALSO

install(cmnd)

BUGS

If the '-s' flag is not specified no messages will be

generated if a specific file is not found in the buildfile.

PROGRAM
c68 - 68000 C compiler

USAGE
c68 [-c] [-[e|f]] [-v] [-L] [-a] [-S] [-[6|7|3|5]] file...

FUNCTION
'c68' is the REGULUS 68000 C compiler. It is identical to the 'cc' command and should be linked (see ln (cmnd)) to '/bin/cc' under REGULUS systems.

For more information on this command, see the documentation under 'cc' (cmnd).

SEE ALSO
cc (cmnd)
as68 (cmnd) which is a link to '/bin/as'
lo68 (cmnd) which is a link to '/bin/ld'

PROGRAM
cat - concatenate files to standard out

USAGE
cat [file ...]

FUNCTION
'cat' concatenates the files specified on the command line and outputs the resulting stream to the standard output. If there are no file names or the name '-' is used, the standard input is read up to end-of-file. Using the '-', files and the standard input may be intermixed, allowing the results of pipelines to be injected into the middle of a set of files.

The output is buffered into 512 byte blocks.

SEE ALSO
list (cmnd), more (cmnd)

BUGS

PROGRAM
cc - 68000 C compiler

USAGE
cc [-c] [-[e|f]] [-v] [-L] [-a] [-S] [-T] [-t] [-[6|7|3|5]] file...

FUNCTION
'cc' is the REGULUS 68000 C compiler. It accepts three types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and loaded all at one go.

The aforementioned flags as well as the C preprocessor flags (eg. -P or -D) See 'cpp(cmnd)' and the link-editor flags (eg. -l or -s) See 'lo68(cmnd)'. The following flags are interpreted by 'cc'.

-c Suppress the loading phase of the compilation and force an object file to be produced, even if only one program is compiled.

-e Specify that floating point arithmetic is used. Instructs the loader to load the floating point library '/lib/libE.a'. Constants will be generated in the 'IEEE Floating Point' format. Constants will be generated in this format by default.

-f Specify that floating point arithmetic is used. Instructs the loader to load the floating point library '/lib/libF.a'. Constants will be generated in the 'Fast Floating Point' format.

-v Force the 'C' source file being processed to be printed regardless of whether there is more than one or not.

-S Compile the named C programs, and leave the assembler-language output on corresponding file suffixed by '.s'. Line number comments are included in the assembly code to help relate the assembly code and the C source.

-L (default) Compile a program which is larger than 32K bytes. This flag instructs the assembler to generate 32 bit addresses for all external references and also changes some code generation sequences (i.e. indexed array references). This flag MUST be specified for all programs whose total size (text size + data size + bss size) is larger than 32K bytes.

-a This is the alternate of the '-L' flag, it causes the assembler to generate 16 bit addresses for all externally referenced variables.

-T Generate assembler code suitable for the 68010. Generates "move cc" instead of "move sr". Let's the assembler know the meaning of the "movec", "moves", "rte", and "rtd" instructions.

-t Causes string constants to be placed in the text segment (as opposed to the data segment). The '-t' flag with an appended 0, 1, 2, or p (eg. '-tp') is used to specify an alternate parser, code generator, assembler or preprocessor respectively. The alternate executables are expected to be located in the directory "/usr/c68".

-6 Version 6 compatibility mode. Adds the v6 include file directory onto the include file search list (/usr/include/v6), and adds the version 6 compatibility library (/lib/libv6.a) to the loader's library search list.

-7 Version 7 compatibility mode. Adds the v7 include file directory onto the include file search list (/usr/include/v7), and adds the version 7 compatibility library (/lib/libv7.a) to the loader's library search list.

-3 System 3 compatibility mode. Adds the s3 include file directory onto the include file search list (/usr/include/sys3), and adds the system 3 compatibility library (/lib/libs3.a) to the loader's library search list.

-5 System 5 compatibility mode. Adds the s5 include file directory onto the include file search list (/usr/include/sys5), and adds the system 5 compatibility library (/lib/libs5.a) to the loader's library search list.

-P Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed '.i'.

-E Run only the macro preprocessor on the named C programs, and write the output to the standard output device. Each code block is identified using a sequence consisting of the source file name and line number (eg. # 34 file.c).

-C Like the -C flag except that the comments are not removed. This is used by lint and similar programs which need information which is stored in comments.

-Dxxx Define the symbol "xxx" as a preprocessor symbol. This flag is useful for setting compile time options such as "DEBUG" or "NOMMU". (See the "ifdef" construct described below)

-Idir Specify a directory (eg. -I/usr/include/test) in which to search for include files which are surrounded by angle brackets <stdio.h> or quotes "ctypes.h" before searching in the default directory '/usr/include'. If '-I' is specified without a directory name the current directory will be used.

Other arguments are taken to be either loader flag arguments, or C-compatible object programs, typically produced by an earlier 'cc' run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name 'a.out'.

The instruction 'asm("text");' will output a single line of text into the assembly output literally.

FILES

file.c	input file
file.o	object file
file.s	assembly language file
file.i	preprocessor output file
a.out	loaded output
/tmp/c6t?a	temporary
/lib/c068	parser
/lib/c168	code generator
/lib/c680.o	runtime startoff
/lib/lib7.a	68000 C library
/lib/libF.a	68000 fast floating point library
/lib/libE.a	68000 IEEE floating point library
/lib/libv6.a	version 6 compatibility library
/lib/libv7.a	version 7 compatibility library
/lib/libs3.a	system 3 compatiblity library
/lib/libs5.a	system 5 compatibility library

SEE ALSO

"The C Programming Language" by Kernighan and Ritchie, published by Prentice-Hall in 1978.
cpp(cmnd), as(cmnd), ld(cmnd), float(misc), a.out(files)
float(misc)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader. To distinguish the source of error messages, compiler error messages begin with an *, assembler error messages begin with an &, and loader error messages begin with a :.

DIFFERENCES

This compiler is an implementation of the language as described in Kernighan and Ritchie Appendix A with the following exceptions and/or machine dependencies:

All pointers are 32 bits in length to accommodate the full 68000 addressing range. Address constants are generated as 32-bit numbers unless the -a flag is specified.

Pointer register variables are assigned to A-registers. All other register variables are assigned to D-registers. There are 5 D-registers and 3 A-registers available for register variables.

Pointer subtraction produces a long result, this only causes trouble when passing the result as an integer argument. A warning is generated.

Structure arguments, and structures returned from procedures are not yet implemented.

An additional comment syntax has been added. Comments may be introduced by two slashes (i.e. //) and are terminated by the end of the source line. The standard C comment syntax (i.e. /* comment */) is also supported.

Unsigned long and unsigned short have not yet been implemented.

PROGRAM
chgrp - change the group of a list of files.

USAGE
chgrp group file1 [file2 ...]

FUNCTION
For each file in its argument list, `chgrp' will change the group-id of that file to be 'group'. The 'group' argument may be either a decimal number or a group name as found in /etc/group.

FILES
/etc/group

DIAGNOSTICS
Zero is returned if all went well, -1 otherwise. Appropriate error diagnostics meant to be self explanatory are printed to standard output.

SEE ALSO

BUGS

PROGRAM

chmod - change mode of a file

USAGE

chmod mode file [file2 ...]

FUNCTION

The mode of each file is changed to the mode specified in the first argument. Only the owner or super user can change the mode of a file. The specified mode can either be absolute or symbolic.

An absolute mode consists of an octal number constructed by ORing the desired bits from the following modes:

04000	set user id on execution.
02000	set group id on execution.
01000	save text (sticky) bit
00400	owner has read permission.
00200	owner has write permission
00100	owner has execute permission
00070	read, write, execute permission for group
00007	read, write, execute permission for others

'chmod 664 file' sets the mode of 'file' to read and write permission for the owner and group, and read for all others.

A symbolic mode specifies changes to the file's current mode. A symbolic mode is constructed as follows:

[who] op what [op what]

'who' is a combination of letters from the set {ugoa}. 'u' specifies the user's permissions, 'g' specifies the group's permissions and 'o' specifies the permissions of others. The letter 'a' is equivalent to 'ugo', the default if 'who' is omitted.

'op' can be + to add permission to the files' mode, - to take permission from the files' mode, or = to assign permission absolutely to the entities specified in the 'who' part of the symbolic mode.

'what' is a combination of letters from the set {strwx}, specifying what permissions are to be added to or subtracted from the current mode. These letters have the following meanings:

s	set owner or group id
t	save text (sticky)
r	read
w	write
x	execute (or search)

's' is ignored for other, 't' can be used with either 'u', 'g', or 'o'. Omitting 'what' can be used with '=' to deny all permissions.

Multiple symbolic modes can be given and are separated by commas. The operations will be performed in the order specified.

'chmod o-rw file' denies read and write permission to others, 'chmod u+s file' makes a file setuid (set user id on execution), and 'chmod o= file' removes all permissions for other from file.

FILES

DIAGNOSTICS

SEE ALSO

chmod (sys)
ls (cmnd)
stat (sub)
chown (cmnd)

BUGS

PROGRAM
chown - change the owner of a file

USAGE
chown owner file ...

FUNCTION
For each file listed in its argument, 'chown' will change the owner of that file to 'owner'. 'Owner' can be a user id or a username. You must be the super user to change the owner of a file.

'chown' will allow you to enter a nonexistent owner id as the owner.

FILES
/etc/passwd

DIAGNOSTICS

SEE ALSO
chown (sub)
ls (cmnd)

BUGS

PROGRAM
chroot - change program's image of directory tree root

USAGE
chroot newroot command

FUNCTION
'chroot' changes a program's image of the directory tree root from slash '/' to something else. Typically used to trap a program into a limited section of the filesystem.

SEE ALSO
chroot (sys)

PROGRAM

cmp - compare two files

USAGE

cmp [-s] [-l] file1 file2

FUNCTION

'cmp' reads two files and compares them one byte at a time. If no flags are specified 'cmp' will generate a message for the first non-matching byte pair. Standard input can be specified by using a '-' symbol in place of file1. If the '-s' flag is specified only an exit value is generated. If the '-l' flag is specified all non-matching bytes will be displayed. This display consists of:

1) The byte offset from the beginning of the files in decimal.
2) The byte value from file1 in octal.
3) The byte value from file2 in octal.

'cmp' prints nothing if the files are identical. If one file is a subsequence of the other, 'cmp' will print "files unequal length".

DIAGNOSTICS

An exit status of 0 corresponds to "files identical", 1 means that the files are different or at least of unequal lengths, 2 means that either the arguments were specified incorrectly or that one of the files could not be accessed.

SEE ALSO

BUGS

PROGRAM
cmpw - compare two files

USAGE
cmpw file1 file2

FUNCTION
'cmpw' reads two files and compares each word (16 bits). All words which do not compare as equal are displayed on the standard output. This display consists of:

1) a byte offset from the beginning of the files in hex
2) the word value from file1 in hex
3) the word value from file2 in hex

If the files are identical, 'cmpw' will print "files identical". If one file is shorter than the other, cmpw will print "files unequal length".

SEE ALSO
size (cmnd)

BUGS

PROGRAM

cp - copy files

USAGE

```
cp [-i] [-l] [-v] file1 file2
cp [-i] [-l] [-v] [-r] dir1 dir2
cp [-i] [-l] [-v] file1... dir
```

FUNCTION

'cp' is the REGULUS general purpose file copy utility. In the simplest case, the first file is copied, and the new copy will have the second name. If the second name is a directory, the file is copied into the directory but will have the same name as the original. Multiple source file arguments may be copied into a directory by giving the destination directory name as the last argument.

As a final case, the entire contents of directories may be copied one from another by giving just the source and destination directory names. If the source file is a directory there can only be one source file, and the destination must be a directory.

The following 2 commands would be equivalent:

```
cp dir/.* dir/* dir2
cp dir dir2
```

The flag arguments to copy are:

-i interactive copy. When a directory source is found, ask the user before copying each of the files in that directory.

-l links. Use links instead of physically copying the files. This is much faster than making physical copies. See 'link (cmnd)' for more on links.

-r recurse. Recursively copy subdirectories and their contents into the destination directory.

-v verbose. Echo to standard output the filenames being copied.

The file protection bits may change if the user's umask is not zero (see umask under sh(cmnd)).

SEE ALSO

cat(cmnd)
link (cmnd)
mv (cmnd)

BUGS

PROGRAM
cpp - 68000 C preprocessor

USAGE
cpp [-C] [-P] [-E] [-D] [-I] [-[6|7|3|5]] source

FUNCTION
'cpp' is the REGULUS 68000 C preprocessor. It may be called specifically or by using 'cc'. The following flags re interpreted by the C preprocessor:

-P Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed '.i'.

-E Run only the macro preprocessor on the named C programs, and write the output to the standard output device. Each code block is identified using a sequence consisting of the source file name and line number (eg. # 34 file.c).

-C Like the -C flag except that the comments are not removed. This is used by lint and similar programs which need information which is stored in comments.

-Idir Specify a directory (eg. -I/usr/include/test) in which to search for include files which are surrounded by angle brackets <stdio.h> or quotes "ctypes.h" before searching in the default directory '/usr/include'. If '-I' is specified without a directory name the current directory will be used.

-6 Version 6 compatibility mode. Adds the v6 include file directory onto the include file search list (/usr/include/v6), and adds the version 6 compatibility library (/lib/libv6.a) to the loader's library search list.

-7 Version 7 compatibility mode. Adds the v7 include file directory onto the include file search list (/usr/include/v7), and adds the version 7 compatibility library (/lib/libv7.a) to the loader's library search list.

-3 System 3 compatibility mode. Adds the s3 include file directory onto the include file search list (/usr/include/sys3), and adds the system 3 compatibility library (/lib/libs3.a) to the loader's library search list.

-5 System 5 compatibility mode. Adds the s5 include file directory onto the include file search list (/usr/include/sys5), and adds the system 5 compatibility library (/lib/libs5.a) to the loader's library search list.

'cpp' provides a macro processing facility with the following commands:

#include "filename"
#include <filename>

Causes the text of the named file to be inserted in the input stream. Files may contain (nested) #include statements. The quoted (single or double) inlude file will first be looked for in the directory where the source file exists which references it, followed by the directories specified using the '-I' flag, and finally in the directory '/usr/include'. An include file surrounded by angle brackets will be looked for only on the include file path list, and in '/usr/include'.

#define NAME string

Causes the contents of "string" to be substituted for NAME whenever NAME is encountered in the input stream. NAME must be a valid "C" identifier.

#define NAME(a,b,c) text

Causes substitution of the text (with parameters) whenever NAME(p1,p2,p3) is encountered in the input stream. NAME, a, b, and c must be valid "C" identifiers.

#undef NAME

Causes the definition of NAME to be forgotten.

#ifdef NAME

If NAME has been previously defined in a #define statement or via the -D flag, include all input until a corresponding #else or #endif is encountered; if NAME is undefined, skip the input. Nesting of conditional statements is permitted. The value of NAME is immaterial.

#ifndef NAME

If NAME has not been defined in a previous #define statement, include all input until a corresponding #else or #endif is encountered in the input stream; if NAME is defined, skip the input.

#if CONSTANT_EXPRESSION

If the CONSTANT_EXPRESSION has a non-zero value, include all input until a corresponding #endif or #else is encountered in the input stream; if CONSTANT_EXPRESSION is zero, skip the input.

#else

If the checked condition in the corresponding #if, #ifdef, or #ifndef is true then any lines between #else and #endif are ignored. If the checked condition is false then any lines between the test and the #else or, lacking a #else, the #endif, are ignored.

#endif

Terminate a #if or #ifdef or #ifndef statement.

#line LINE [FILENAME]

Coerces the preprocessor and the parser into believing that the next line is in the specified file on the specified line. If no filename is specified uses the current source file name.

There are two predefined, user redefineable macro's __FILE and __LINE while will respectively be evaluated to the current file name enclosed in double quotes and line number respectively by the preprocessor.

FILES

file.c	input file
file.s	assembly language input file
file.i	preprocessor output file

SEE ALSO
cc(cmnd)

PROGRAM
date - print the current date and time

USAGE
date [mm/dd[/yy]] [hh:mm[:ss]]

FUNCTION
'date' prints the current date and time.

The superuser can change the current date and time using date. The format for date is mm/dd[/yy] (the year is optional), the format for time is hh:mm[:ss] (seconds are optional).

FILES
none

SEE ALSO
time(sys)

BUGS
Must be set-uid-root.

PROGRAM
dd - convert and copy a file

USAGE
dd [option=value]...

FUNCTION
'dd' reads from the designated file or from standard input, carries out the specified conversions and copies the file to the designated output file or standard output. Two types of options are available: file and buffer specifications and types of conversions.

if=f	Input file name. If not specified standard input will be used.
of=f	Output file name. If not specified standard output will be used.
ibs=s	Input buffer size. The default is 512 bytes.
obs=s	Output buffer size. The default is 512 bytes.
bs=s	Buffer size. Set input and output buffers to the specified size.
cbs=s	Conversion buffer size. For use with ASCII or EBCDIC conversion formats.
skip=n	Skip over the first n input records.
count=n	Copy only the specified number of input buffers.
conv=ascii	Convert EBCDIC to ASCII.
conv=ebcdic	Convert ASCII to EBCDIC.
conv=lcase	Convert all upper case alphabetics to lower case.
conv=ucase	Convert all lower case alphabetics to upper case.
conv=swab	Byte swap every pair of bytes in the input buffer.
conv=noerror	Don't stop processing on an error.
conv=sync	Pad partial input buffers to ibs.

More than one conversion can be specified at a time if they are separated by commas.

All numbers will be interpreted as decimal unless followed by a 'k', 'b' or 'w' specifying multiples of 1024, 512 and 2 respectively. The product of two numbers (separated by an 'x') will also be accepted.

For example, to byte swap and convert a file xxx to all lower case file yyy using 128 word buffers:

```
dd if=xxx of=yyy bs=128w conv=swab,lcase
```

DIAGNOSTICS
'dd' prints the number of complete and partial records

which were read and written.

SEE ALSO
cp (CMND)

BUGS

PROGRAM
ddt - dynamic 68000 object code debugger

USAGE
ddt [a.out [core]] [-]

FUNCTION
'ddt' is a debugger for use with C and assembly language programs. It is useful for both post-mortem and interactive debugging. An important feature of 'ddt' is that even in the interactive case no advance planning is necessary to use it; in particular it is not necessary to compile or load the program in any special way nor to include any special routines in the object file.

The first argument is the object program (containing a symbol table); if not given, 'a.out' is used. If only one argument is given, it is assumed to be an object program. The second argument is a core-image file; if not given, 'core' is used. The core file need not be present.

The optional third argument specifies that the file being debugged is not a core image. This feature is used most frequently in examining the memory file /dev/mem.

In order to set breakpoints and execute, the namelist (a.out) file must be present and can not be a shared text program.

Commands to 'ddt' have the following form:

[<address expression>[,repetition]]<command>[<display mode>]

If no address is given, the last displayed address is used (this value is accessible by typing the special symbol '.'). An address expression may optionally be followed by a comma and a number, in which case the command applies to the appropriate number of successive addresses.

<address expression>

Address expressions consist of any combination of operator-separated symbol names, decimal numbers, octal numbers which begin with '0', hexadecimal numbers which begin with '$', single ASCII character constants which begin with ''', and double ASCII character constants which begin with '"'. There is no provision for input of floating point numbers.

The following operators are recognized:

+	addition
blank	addition
-	subtraction
*	multiplication

\	division
%	mod (remainder)
|	OR
&	AND
@	indirection
[]	array subscription

Evaluation proceeds left-to-right. The construction 'name[expression]' assumes that name is a pointer to an integer and is equivalent to the contents of the named cell plus twice the expression.

<command>

The command strings are:

/ Display the contents of the addressed memory locations according to the current display mode (default 'i').

= Evaluate and display the address expression according to the current display mode (default 'o').

! Store the address expression (16 bits) into the specified memory word (default is dot). Before the execution of a ;g command (ie. before creation of the child task), all store commands (!, #, `) actually modify the task image which is stored on the disk file. After a ;g command, these store commands modify only the child task image in memory.

` Store the 32 bits of the address expression into the specified long word (default is dot).

Store the lower 8 bits of the address expression into the specified byte (default is dot).

! If no address expression precedes the '!', the remainder of the line after the '!' is sent to REGULUS to be interpreted as a command.

? Display fault type.

<new line> Display the next memory location according to the current display mode.

^ Display the previous memory location according to the current display mode.

> Replace dot with the effective address of the last instruction which was displayed (follow the path). For two operand instructions, the effective address of the first operand (source)

is used.

;= Search equal.

;# Search not equal.

;a Effective address search. Beginning at the last address displayed, search sequentially for an instruction with the specified effective address. When the first one is found, stop the search and print the instruction.

;b Set an execution breakpoint.

;c Clear an execution breakpoint.

;ca Clear all execution breakpoints.

;d Display the location of all execution breakpoints.

;e Display the floating point registers.

;f Exit from 'ddt'.

;g Begin execution of the symbolic file at location 0. This command may be optionally followed by a list of character string arguments to the file.

;i Continue execution without passing any signal to the child process. This command is most useful after hitting sending an interupt to 'ddt'. ';p' passes the signal to the child process, while ';i' does not. Otherwise their actions are identical.

;l List symbols from the symbol table. The command is followed by a space and a generic symbol name pattern. The symbol table is searched and all symbols which match the pattern are displayed along with their address and symbol type. The character '?' in the pattern matches any single character in the symbol name; the character '*' in the pattern matches any number (including 0) of characters in the symbol name. For example, the command ';l m*' will list all symbols that begin with the letter 'm'. The command ';l *' will list all symbols in the symbol table.

;p Continue execution after a breakpoint.

;r Display all registers in octal and as symbols.

;s Execute one instruction and break.

n;s Execute 'n' instructions and break.

;t Print a stack trace of the terminated program. The calls are listed in the order made; the actual arguments to each routine are given in octal.

;. Set a temporary breakpoint at the specified address and continue execution. This temporary breakpoint is deleted when the next breakpoint is encountered.

<display mode>

The mode characters are:

o	octal (word)
x	hexadecimal (word)
d	decimal (word)
l	unsigned decimal (range 0 to 65535)
f	floating point
e	double precision floating point
i	symbolic instruction
&	address
b	octal (byte)
h	hexadecimal (byte)
t	decimal (byte)
'	byte in ASCII
"	word in ASCII
s	string in ASCII

There are two display modes, one for display contents (/) and one for evaluate (=). When 'ddt' is invoked, the default mode for display contents is 'i' (instruction) and the default mode for evaluate is 'o' (octal). Whenever an explicit display mode is given, the current mode is set to the given mode and used thereafter until another mode is given.

<special symbols>

'ddt' recognizes a number of special symbols. The values of these special symbols may be displayed, modified, and used in expressions.

name	meaning	default value
:d0	user register D0	
:d1	user register D1	
:d2	user register D2	
:d3	user register D3	
:d4	user register D4	
:d5	user register D5	
:d6	user register D6	
:d7	user register D7	
:a0	user register A0	

:a1	user register A1	
:a2	user register A2	
:a3	user register A3	
:a4	user register A4	
:a5	user register A5	
:a6	user register A6	
:d5	user register D5	
:sp	user stack pointer	
:pc	user program counter	
:ps	user processor status word	
:m	search mask	0177777
:lo	lower search limit	0
:hi	upper search limit	020000
:l	lowest address to display as a symbol	0100
:q	last value typed to user	
:d	D space symbol offset 0	
:t	text (I) space symbol offset	0
:il	ignore illegal instructions	0
:ib	radix for inputting numbers	10
:a	0 -> only C symbols 1 -> C and assembler symbols	0
:r	maximum symbol offset to display symbolically	01000
:f	current function for local variables	last function called
:sl	lowest address on stack	0177000
:cp	current subroutine return address	

The swapable per-task area (s_tasktab) of the child task is available for inspection by examining the addresses between -2048 (beginning) and -2 (end). This area may not be modified.

Examples (each command is terminated by a <new line>):

t1/	Display the contents of t1 in the current display mode.
t1=	Display the address of t1 in the current evaluate display mode.
t1+40/d	Display the contents of 't1+40' (decimal) in decimal.
@p/o	Displays in octal the contents of the word pointed to by 'p'.
:sp=o	Display the contents of the users stack pointer (r6) in octal.
0377!:d2	Set the value of the users register d2 to 0377 octal.

string/s	Display the contents of string as ASCII characters.
xyz+10/o	Display in octal.
xyz+012 023147	'ddt' displays the cell value.
x+012 010 027!	User changes 'x+012' from octal 10 to octal 27.

Searches:

0100!:lo	Set lower search limit to 100 octal.
1000!:hi	Set upper search limit to 1000 decimal.
0177400!:m	Set search mask to upper byte.
023000;=	Search between 100 octal and 1000 decimal and print the address and contents of each word if the value of that word when logically ANDed with octal 177400 is equal to octal 23000. A search not equal also searches between the limits in ':8' and ':9', ANDs each word with the mask in ':m' and displays each word for which the resulting value does not equal the given value.
xyz;a	Beginning with the last address displayed (dot), search sequentially for the first instruction which has an effective address of xyz. When one is found, terminate the search and print the instruction. For double operand instructions, both effective addresses are compared.

Execution:

The general format of the set breakpoint command is:

<address expression>;b[<conditional expression>]

which causes a breakpoint to be installed at location <address expression>. When a conditional breakpoint is encountered during the course of execution, the conditional expression is evaluated. If the value of the conditional expression is true (non-zero), the execution break occurs. If the value is false (zero), the execution break does not occur, and execution continues in the normal fashion. A null conditional expression is always evaluated as true.

The format of the conditional expression is:

<relational expression>[<boolean operator><relational expression>

<boolean operator><relational expression>...]

A conditional expression is thus composed of any number of relational expressions separated by the boolean operator (&& for AND) or (|| for OR). Evaluation is strictly left to right (no parentheses).

A relational expression is an address expression or two address expressions separated by one of the relational operators: ==,!=,<,<=,>,>=. If an address expression contains a symbol name then the value of the address expression is taken to be the contents of the memory cell addressed by the expression. If an address expression contains no symbol name then the expression is taken literally (i.e. the value of 'il+4' is the contents of the memory cell whose address is 'il+4' while the value of 100+4 is 104). The expression '@104' has the value of the contents of memory cell 104. The expression '@il+4' has the value of the contents of memory cell 'il' plus the value 4.

CAUTION: blanks are meaningful to 'ddt' and are equivalent to the addition (+) operator.

The command ';d' displays each active breakpoint along with any conditional expression associated with each.

Some valid conditional expressions are:

```
xx
il>=0100
funl:argl!=1&&ii<12
jl||j2
@0177203==0177777
```

Upon encountering a breakpoint, 'ddt' displays the instruction which will be executed when execution continues and then goes into the command mode.

Examples:

;g abc filea	Starts execution of the symbolic (a.out) file at location 0 passing to it the arguments 'abc' and 'filea'.
;s	Execute the next instruction and break.
5;s	Execute the next 5 instructions and break.
funl;.	Set a temporary breakpoint at 'funl' and continue execution.
xxx;c	Clear the breakpoint at location 'xxx'.
;ca	Clear all breakpoints.

;g <f1 >f2 This command redirects the standard input and output files just like the corresponding shell command.

Symbols:

All C symbols begin with the underscore character '_'. As long as the special symbol ':a' contains a value of zero (default), 'ddt' prepends an '_' to all symbols, thus eliminating the requirement for the user to type '_'. If ':a' is set to a nonzero value, indicating that the program contains symbols which did not pass through the 'c68' compiler (such as assembler symbols), the '_' before each C symbol will be displayed by 'ddt' and must be typed by the user when specifying a symbol name.

Variables local to a function (automatic, static, and arguments) are accessible by writing the function name followed by a colon ':' followed by the local variable name (i.e. main:argc). The notion of the current function exists and is addressable as the special symbol ':f'. The value of ':f' is set to the last function to be invoked whenever a breakpoint is encountered. Variables local to the current function are accessible by simply writing the variable name. The current function can be changed by modifying ':f' (i.e. xyz!:f sets the current function to xyz which must be active).

DIAGNOSTICS

Self-explanatory diagnostics are given for illegal or unrecognized commands.

SEE ALSO

c68 (cmnd)
C Reference Manual

BUGS

Evaluation of conditional expressions on breakpoints are not yet implemented on the 68000 -- all breakpoints are unconditional.

PROGRAM
df - disk free space summary

USAGE
df [filesystem]

FUNCTION
'df' gives a free space summary for a filesystem. The summary contains the number of blocks used, the number of free blocks and the percentage of the total space used. If a filesystem is not specified 'df' uses a set of default filesystems found in the file '/etc/fstab'. The '/dev/' prefix is optional in the filesystem specification.

FILES
/etc/fstab - default filesystems
/etc/mtab - mounted filesystems table

SEE ALSO
fsck (cmnd)
fstab (files)
mtab (files)

BUGS

PROGRAM

dir - directory or archive information

USAGE

dir [filename] ...

FUNCTION

If the argument is a directory, dir first prints the word "Directory" followed by the full pathname of the directory. If the file ".dirname" appears in the directory, its contents are then printed. This allows printing of helpful information about the contents of a directory. This is then followed by a sorted columnar listing of the contents of the directory. If no information can be gotten about a file in a directory because the `stat' system call failed, `(?)' is printed before the file name. If the file in the directory is itself a directory, `[]' surrounds the name. If the file is a character special file, or a block special file, `(*)' appears before the file name. If the file in the directory is executable, `{}' appears around the name. A plain file appears alone.

If the argument is an archive file, the word "Archive" is printed, followed by the full pathname of the archive file, followed by the sorted columnar listing of the contents of the archive.

if the argument is another type of file, "dir" will attempt to determine its type and print an appropriate message.

If no filename argument is given, dir lists the contents of `.'.

FILES

.dirname

DIAGNOSTICS

SEE ALSO

pwd(cmnd), ls(cmnd), stat(sys)

BUGS

PROGRAM
dsw - interactive delete

USAGE
dsw [file ...]

FUNCTION
'dsw' removes files interactively. If a directory is given it will prompt the user for all the files in the given directory. If no files are given 'dsw' defaults to the current working directory. Each filename in the directory is printed followed by a question mark. Typing a 'y' will delete the file. Typing an 'x' will cause 'dsw' will halt. Any other character is considered a negative response and the file is not deleted.

SEE ALSO
rmdir (cmnd), rm (cmnd)

BUGS

PROGRAM
du - disk usage.

USAGE
du [-s] [-a] [file]

FUNCTION
'du' reports the number of blocks in the directory or file named and recursively through lower directories and files. If the file is not specified 'du' acts on the current directory. The following flags are allowed:

-s summarize
-a all files

'du' lists all directories which it encounters and their block sizes unless one of the options is specified. If the -a option is used all files will be listed along with their block sizes. If the -s option is used only the total number of blocks will be reported.

FILES
. default directory name

DIAGNOSTICS

SEE ALSO
df (cmnd)

BUGS
Needs an 8K stack.

PROGRAM
echo - echo arguments

USAGE
echo [-n] [arg ...]

FUNCTION
'echo' outputs its command line arguments to the standard output separated by spaces. the line is terminated by a carriage return unless the '-n' argument present. It is primarily used with command files and pipes.

BUGS

PROGRAM
ed - line oriented text editor

USAGE
ed [-] [+] [-t#] [filename ...]

FUNCTION
'ed' is a line oriented text editor.

'ed' commands always take affect on a special area called a buffer. This buffer goes away before 'ed' exits. If it is desired for the contents of this buffer to be put into a file, then the buffer must be written to a file with the w command.

'ed' has three optional arguments. The first two are useful when 'ed' is used in a shell script. The minus ('-') tells 'ed' not to print out line numbers, command prompts, or character counts for the r and w commands.

The plus argument ('+') tells 'ed' not to print out a warning message if the buffer has not been written to a file since the last time it was changed, and you are leaving the editor.

If the argument '-t' is given followed by a number, the default page length for page commands is set to the number given. This page length will stay in effect until it is changed by one of the page commands.

If filename arguments are given, the names are stored for use with the 'next file' command (see below). The first filename is then read from the list and if a file with this name exists, the buffer will initially contain a copy of this file. If the file does not exist, the buffer will be empty. In both cases the default filename for various commands will become 'name'.

'ed' commands are comprised of up to three possible parts; an address part, a command part, and a parameter part. For those 'ed' commands which do not operate on lines, an address part is not required. The command part is a single letter command, and the parameter part is a modifier to the command. Some commands do not require or allow a parameter. For commands that require addresses, there are default addresses, so that the address part may be left out.

'ed' is always in one of two possible modes: input mode or command mode. 'ed' starts out in command mode. Unless otherwise noted, 'ed' will not allow more than one command on a line.

Commands which put you in the input mode collect text typed in at the terminal, and place it in the buffer according to the address part of the command. In order to get back to

the command mode, you must type a '.' followed by a carriage return at the beginning of a line.

'ed' uses a limited form of a powerful notation known as regular expressions (RE). A regular expression is a string that can specify a (possibly) larger set of strings. The regular expression is said to 'match' each member of the larger set of strings.

There are some characters in 'ed' which have more than their normal meaning when used in regular expressions. These characters are called metacharacters. The set of metacharacters is shown below:

^ ~ $? * [ ]

In order to prevent their special meaning in regular expressions, each metacharacter must be preceeded by the '\' character. This is also true of the '\' itself, and the characters which bound the regular expression. The set of regular expressions allowed by 'ed' is formed by using the following constructs:

1. A normal character (not a metacharacter) is a regular expression and matches that character.

2. A circumflex '^' or tilde '~' at the beginning of a regular expression matches the empty string at the beginning of a line.

3. A dollar sign '$' at the end of a regular expression matches the empty string at the end of a line.

4. The question mark '?' matches any single character except the new-line character.

5. A regular expression followed by an asterisk '*' matches any number (including zero) of adjacent occurrences of the regular expression.

6. Any string of characters between a left bracket '[' and right bracket ']' matches any one occurrence of a character in that string. If the first character of the enclosed string is a circumflex or tilde, the regular expression matches any character except the new-line and the characters in the string. A range of characters may be specified by giving the characters that represent the bounds of the range separated by a dash '-'.

7. The concatenation of two regular expressions is a regular expression that matches all strings which are the concatenation of the strings matched by the first regular expression followed by the strings matched from the second regular expression.

8. The empty or null regular expression is equivalent to the last regular expression formed.

As mentioned before, many 'ed' commands use an address part to denote the lines over which the command will take effect. When explicitly used, these addresses can take two forms. One is to use a number which represents the lines themselves. Another is to use a regular expression for any or all addresses a command might use.

In ed, there is always a 'current' line. The line number of the current line is displayed in the command prompt as "nnn>", where 'nnn' is the line number. Unless otherwise noted in the discussion of the commands below, the current line is made to be the last line affected by a command.

The address part of all 'ed' commands that require an address is formed by using the following conventions.

1. The character '.' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A base ten number n addresses the n-th line of the buffer.

4. A regular expression which begins with the character '/' addresses the first line found that contains a string matched by the regular expression. The search extends from the current line toward the end of the buffer and if necessary wraps around to the beginning of the buffer. The RE may end with another '/' or a carriage return. A '/' followed by a carriage return stands for the null RE and will match all strings specified by the last RE.

5. A RE beginning with a '?' also addresses the first line found that contains a string matched by the RE. The search extends from the current line toward the beginning of the buffer and if necessary wraps around to the end of the buffer. The RE may be terminated by either another '?' or a carriage return. A '?' followed by a carriage return is equivalent to two '?'s and matches the strings matched by the previous RE.

6. An address followed by a '+' or '-' followed by a decimal number specifies that address plus (or minus) the indicated number of lines.

7. If an address begins with '+' or '-', the addition or subtraction is taken with respect to the current line; e.g. '-5' stands for '.-5'.

8. If an address ends with '+' or '-', then 1 is added (or

subtracted). As a consequence of this rule and rule 7, the address '-' refers to the line before the current line and the address '+' refers to the line after the current line. Trailing '+'s and '-'s have a cumulative effect, so that '++' refers to the current line plus 2.

9. To maintain compatibility with earlier versions of the editor, the character '^' in an address is equivalent to '-'.

If an address is given for a command that does not require one, it is an error. It is also an error for a RE to not match any lines. If one or more of the required addresses is missing, the command will use its default addresses (see below). If more addresses are given than the command requires, the last one(s) given will be used.

A ',' is normally used to separate the addresses for a multiple address command. If a ';' is used as a separator, the current line is set to the address preceeding the ';' before the next address is interpreted. This feature is used to denote the starting point for forward and backward searches ('/', '?'). The second address of any two-address command must specify a line following the line specified by the first address. When ';' is used, an unspecified address defaults to '$'.

In the following list of 'ed' commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below.

'ed' issues a command prompt, "nnn>", whenever it is ready to accept a new command.

(.) a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

(. , .) c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is

left at the last line input; if there were none, it is left at the first line not deleted.

(. , .) d
The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

(. , .) e
The edit command causes the addressed lines to be displayed and opened for line editing. All functions of the line editor are available. The specified line in the file becomes the "old line"; to copy a line unchanged, use the CTRL D command. See the line editor summary 'ledit (misc)' for more on the line editor.

f
f filename
The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1 , $) g/regular expression/command list
(1 , $) G/regular expression/command list
This is the global command. For each line that is matched by the RE, this line becomes the current line '.', and then the command list is executed. The command list is comprised of one or more 'ed' commands except another global command ('g' or 'v'). Only one command may appear on a line, so every line except the last in a multiple command list must have a '\' as the last character of the line. If the last command of a list is one which places you in input mode ('i', 'a', or 'c'), the '.' terminator for that command may be omitted.

If a upper case 'G' is used as the command, for each line that contains a string that matches the RE, the editor will print the line, and prompt for input by printing a '?'. A response of 'y' will cause the command list to be performed on that line. A response of 'a' causes the editor to perform the command list on the matched lines remaining without prompting. A 'q' will stop the command at the current line and return to command mode. Any other response causes the editor to skip that line.

(.) i
<text>
.
This command inserts the given text before the addressed line. '.' is left at the last line input; if there were none, at the addressed line. This command

differs from the 'a' command only in the placement of the text.

(. , .+1) j
This command concatenates or joins lines together. The addressed lines are replaced with the result of concatenating them together. The join command tries to be reasonable and places a space between the end of one line and the beginning of the next. If this is not desired the space must be removed by the user (see the substitute command below).

(. , .) l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in octal, and long lines are folded. An l command may follow any other on the same line. The tab character is printed as '>' and backspace character as '<'. '.' is left at the last line listed.

(. , .) ma
The move command repositions the addressed lines after the line addressed by a. The last of the moved lines becomes the current line.

n
n filename
The n command is used to read a new file into the buffer. The previous contents of the buffer are lost. You will be warned if the buffer has changed and no write has been done, unless the '+' flag was used when the editor was started. To ignore the warning, issue the command a second time.

In the first form, the next file from the list of filenames given the editor when it is started is read into the buffer. The name of the file is printed to assist the user. If the list is empty, no file is read in but the previous contents of the buffer are lost anyway.

In the second form, the file with the name given is loaded into the buffer. This does not affect the contents of the initial filename list. The number of characters and lines read will be printed, and the current line will be set to the last line read.

The remembered filename becomes the name of the file that was read. This command is often used instead of quitting the editor and restarting with a new file since the editor load time is reduced.

(. , .) p
The print command prints the addressed lines. '.' is

left at the last line printed. The p command may be placed on the same line after any command.

q
Q

The quit command causes ed to exit. No automatic write of a file is done. If any modifications have been made to the file since the last 'w' command was issued, the 'q' command is considered erroneous and a warning message "Write???" is issued. To quit without writing, issue 'q' a second time. You may force a quit without warnings by invoking the editor with the optional '+' argument or by using a capital 'Q'. A 'Q' will always force the editor to exit.

(.) r filename

The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see the f command). The remembered file name is not changed unless 'filename' is the very first file name mentioned. Address '0' is legal for r and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file.

(. , .) s/regular expression/replacement/ or,
(. , .) s/regular expression/replacement/g or,
(. , .) s/regular expression/replacement

For each line specified by the address part of the 's' command, a string matching the regular expression (RE) is searched for. The first occurrence of the matched string is then replaced by the replacement string. If the global indicator 'g' is present, every occurrence of the matched string in the line will be replaced by the replacement string. It is an error for the RE to fail to match any string in the address range of the 's' command.

Any printing character, with the exception of the space or new-line, can be used in place of the '/' to surround the RE and its replacement string. The current line becomes the last line for which a match was found for the RE.

The '&' character has special meaning when used in the replacement string. 'ed' replaces it with a copy of the string matched by the RE. To prevent this, you must precede the '&' by the '\'. In order to split a line into 2 or more lines, a new-line may be used in the replacement string if it is preceeded by a '\'.

If the final separator is omitted, the last line containing a string matched by the RE will be printed

after the replacement string has been substituted. If no RE and no replacement string are specified, the last substitution is performed again. The resulting line will be printed.

(. , .) t a
This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0). '.' is left on the last line of the copy.

(1,$) v/regular expression/command list
(1,$) V/regular expression/command list
This command has the same effect as the 'g' command, only the command list is performed only on those lines which do not contain a string which matches the RE. The 'v' command causes prompting just like the 'g' command.

(1,$) w filename
(1,$) wq filename
The write command writes the addressed lines onto the given file, destroying its present contents (if any). If the file does not exist, it is created mode 666. The access mode of an existing file is retained on writing; access modes may be changed with the 'chmod (cmnd)' command. The remembered file name is not changed unless 'filename' is the very first file name mentioned. If no file name is given, the remembered file name, if any, is used (see the f command). '.' is left at the last line written. If the command is successful, the number of characters and lines written is typed. The variant form 'wq' causes the file to be written and the editor to exit at once.

(1,$) z filename
(1,$) zq filename
The z command is used to append the addressed lines at the end of an existing file. If the file does not exist, it is created with mode 666. If no file name is given, the remembered file name is used. '.' is left at the last line written.

(.) P
The 'P' command will turn prompting on or off. This is a toggle switch. The opposite of the current setting will be restored.

(.+1): n
Print the next 'n' lines beginning at the addressed line. The default value of 'n' is 20. If a value for 'n' is specified, that value will become the new default for all page commands until it is changed again.

(.)@ n
Print a page of 'n' lines with the specified line in the middle of the page. The default value of 'n' is 20. If a value for 'n' is specified, that value will become the new default for all page commands until it is changed again.

(.)* n
Print a page of 'n' lines with the specified line at the bottom. The default value of 'n' is 20. If a value for 'n' is specified, that value will become the new default for all page commands until it is changed again.

(.)_ n
Print a page of 'n' lines starting 2*n lines from the specified line. This effectively means the page 2 pages back. The default value of 'n' is 20. If a value for 'n' is specified, that value will become the new default for all page commands until it is changed again.

($)=
The line number of the addressed line is typed. '.' is unchanged by this command.

!REGULUS command
The remainder of the line after the '!' is sent to REGULUS to be interpreted as a command. '.' is unchanged.

(.+1 , .+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text.

If an interrupt signal (normally ASCII DEL) is sent, ed prints a '?' and returns to its command level.

Some size limitations: 128 characters per line, 256 characters per global command list, 100 characters per file name. The limit on the number of lines depends on the amount of core: each line takes 14 bytes.

FILES
/tmp/E#, temporary;
'#' is the process number (in octal).

DIAGNOSTICS
'?' for errors in commands; 'TMP' for temporary file overflow.

SEE ALSO

BUGS

PROGRAM
env - execute task with specified environment

USAGE
env [-] [name=value ...] [prog arg ...]

FUNCTION
'env' is used to execute tasks with a given environment. The environment passed to the task is constructed from env's inherited environment plus any arguments of the form: name=value. If the '-' flag is present, then the inherited environment is ignored. If no task is specified, then the environment env has created is printed out one environment variable per line.

SEE ALSO
getenv (subs)
environment (misc)

BUGS

PROGRAM

expand - expand tabs to spaces

USAGE

expand [-#] [-#,#...] [file...]

FUNCTION

Expand replaces all tabs with spaces. The default tab size is every 4 columns. If a different tab size is preferred it can be specified using the '-#' option. This will expand the tabs to every '#' columns.

If individually specified tab stops are preferred they may be specified using the '-#,#...' option, in which the desired tab stops are separated by commas. The actual column numbers where the tab stops are to be placed are given. If there are more tabs than tab stops specified, the default spacing is used.

The result is printed on the standard output.

SEE ALSO

list (cmnd)

BUGS

PROGRAM
file - determine file type

USAGE
file file1...

FUNCTION
'file' tries to determine the type of each named file. Among the types recognized are: a.out, roff, archive, 'c', text and data files. Special files and directories are also recognized.

SEE ALSO

BUGS

PROGRAM
find - find files in the filesystem

USAGE
find [pathlist] expression...

FUNCTION
'find' looks for files which fit the specified conditions according to the given boolean expression. 'find' looks for files starting from the pathname given recursively through directories. If no pathname given, the search starts from the current directory.

The boolean expression is formed from the following primitives:

-name f Evaluates true if the name matches 'f'. The specified filename may contain the standard shell filename matching characters (asterisks, question marks and brackets) for pattern matching. If the name contains these special characters, it must be quoted to prevent the shell from attempting to expand the name.

-perm o Evaluates true if the file's permissions match the specified octal number. If the number is preceded by a '-' then the entire set of permissions (017777) is considered, otherwise only the first 9 bits (0777) are considered.

-type c Evaluates true if the file is of the specified type. The types are: 'b' block special, 'c' character special, 'd' directory, 'p' named pipe, or 'f' plain file.

-links n Evaluates true if the file has 'n' links to it.

-user u Evaluates true if the file belongs to the specified user. The user's login name or numeric user id may be specified.

-group g Evaluates true if the file belongs to a user in the specified group. The group must be specified with the numeric group id.

-size b Evaluates true if the file is 'b' blocks in size. If the 'b' is preceded by a '+' then it evaluates true if it is greater than or equal to the size, and if it is preceded by a '-' then it evaluates true if the size is less than or equal to the specified size.

-atime n Evaluates true if the file has been accessed within the last 'n' days.

-mtime n Evaluates true if the file has been modified within the last 'n' days.

-newer f Evaluates true if the file's modification date is greater than the file 'f'.

-print This expression always evaluates to true. If the entire expression to this point has evaluated to true the current full pathname will be printed.

-inum n Evaluates true if the current file's index record number is n.

-exec cmd Evaluates true if the executed command returns a zero error status. A pair of double curly braces '{}' will be replaced by the current full pathname.

-ok cmd Exactly like 'exec' except that the user will be prompted with the generated command before it is executed. If the user responds with 'y', the command will be executed. Any other character will be considered a negative response.

One or more conditions may be specified at a time. Multiple expressions can either be separated by boolean operators or an implied 'must also be true' will be in effect. The expressions are evaluated in a strictly left to right manner. The three following operators are understood.

-a 'and' operator. The expressions to either side of the operator must evaluate true.

-o 'or' operator. If either condition is true the expression evaluates to true.

! 'negation' operator. The 'not' operator changes the value of the expression that follows.

An example of 'find' would be to search for all the core files which haven't been accessed in three days and remove them.

```
find -name core -a ! -atime 3 -exec "/bin/rm {}"
```

Another example would be to list the path name of all files that are larger than 50 blocks.

```
find -type f -size +50 -print
```

FILES
/etc/passwd

SEE ALSO

BUGS
Only one of each type of expression can be used.

PROGRAM
grep - regular expression pattern search

USAGE
grep [-v] [-c] [-n] pattern [file ...]

FUNCTION
'grep' searches for a pattern in the given file or files. The pattern is a "regular expression" as used in the editor, which can include special characters such as *, ?, [and]. If no file is specified, the standard input is read and searched.

The following flag arguments are understood:

-v find non-matching lines
-c print a count of the number of matching lines
-n also print the line numbers of matching lines

Unless the -v or -c options are specified 'grep' will print all lines which match the pattern.

If the -n option is used 'grep' will print the line number of the matching lines before the line.

If the -v option is used only non-matching lines will be printed.

If the -c option is used only a count of the number of matching or non-matching lines (depending on the -v option) will be printed.

SEE ALSO
ed (cmnd) for a description of regular expressions

BUGS

PROGRAM
head - display the first lines of a file

USAGE
head [-#] [file...]

FUNCTION
'head' displays the first few lines of the specified files. It will print out ten lines unless otherwise specified. If a file is not specified on the command line 'head' reads from the standard input.

SEE ALSO
tail(cmnd)

BUGS

PROGRAM
help - display REGULUS documentation

USAGE
help [-p] [-k] command [class]

FUNCTION
'help' displays the documentation for the specified item. If no command is given, 'help' will display the documentation on 'help'. This is the same as 'help help'.

Documentation is available for each of the sections from the REGULUS manual. Each section is searched in order, and the first occurrence of the item is then shown.

To specify a particular manual section, use the 'class' option. This is used where commands or subroutines have the same name. The classes are: cmnd, sys, subs, files, stdio, v6, v7, sys3, dev, misc and maint. 'v6', 'v7', and 'sys3' are the documents for the compatibility subroutines and system calls.

'help' normally displays the first page of the documentation as soon as it is ready. If the optional argument '-p' is used, 'help' will pause before printing the first page.

If no document is found that matches the command given, 'help' will search the permuted index for a matching line. This can allow the user more topics to look for.

The optional argument '-k' is used to only search the permuted index.

FILES
/doc/man/[class]
/doc/man/perm.idx Permuted index

SEE ALSO
man (cmnd)

BUGS

PROGRAM
install - install a file in a directory

USAGE
install [-c dir] [-f dir] [-i] [-n dir]
[-o] [-q] [-s] file [dir1 dir2 ...]

FUNCTION
'install' is used to install files in appropriate directories. If 'install' is called without any flags, it will check for the existence of 'file' using the find(cmnd) command.

If 'file' is not found in any of the directories that are on the command line, then 'install' will search for 'file' in its list of default directories in this order: /bin, /usr/bin, /etc, /lib, /usr/lib. If 'file' is not found in any of these directories either, 'install' will report this and exit immediately. If 'file' is found in one of the directories, 'install' will copy 'file', using cp(cmnd), to the first directory in which it was found.

There are many possible flags to the 'install' command:

-c dir Used for installing brand new commands in the directory following the '-c' flag. If 'file' does NOT already exist in the directory specified, it will be copied to this directory. However, if 'file' IS found in this directory, 'install' reports this and exits WITHOUT copying it to this directory.

-f dir Forces 'file' to be installed in the directory following the '-f' flag, regardless of whether or not 'file' was found in this directory. If 'file' did not already exist in this directory, the owner is changed to 'bin' (using chown(cmnd)), and the mode is changed to 0755 (using chmod(sys)). If 'file' already existed, the owner and mode will remain the same.

-i Causes 'install' to ignore its default directories. It will search only through the directories specified on the command line. This flag may not be used with the '-c' and '-f' flags.

-n dir 'Install' will copy 'file' into the directory specified after the '-n' flag only if it is not found in any of the directories it has searched. If 'file' already existed in the directory following the '-n' flag, it will retain its original mode and owner. Otherwise, its owner will be changed to 'bin', and its mode set to 0755.

-o If 'file' is found in one of the directories,

'install' will first save a copy of the old 'file' by renaming it 'OLDfile', using mv(cmnd).

-q Search quickly: Instead of using find(cmnd) to search for 'file', 'install' will do a 'stat(sys)' in the directories it searches. This makes 'install' significantly faster, but causes it to search only at the top level of the directories. However, this is usually sufficient.

-s Causes 'install' to suppress reporting anything except for errors.

SEE ALSO

find(cmnd)
chown(cmnd)
chmod(sys)

BUGS

PROGRAM
kill - kill a process

USAGE
kill [-#] pid ...

FUNCTION
'kill' will stop the processes with the given process id's. 'kill' by default sends a terminate signal to the specified processes. If a legitimate signal is designated preceded by a minus sign as the first argument, 'kill' will substitute that signal for the terminate signal.

A pid of a positive number will result in the signal being sent to that specific process. A pid with a process group id preceded by 'g' will send the signal to all of the processes in the specified process group. A pid of "group" will send the signal to all of the processes in the current process group. This essentially kills all tasks associated with the login shell, and logs the current user out.

A pid of "all" will send the signal to all of the processes whose real user id matches the current effective user id. This essentially logs the current user out of everywhere he is logged in. If the super user does a "kill all", this will log out all users.

DIAGNOSTICS

SEE ALSO
ps (cmnd)
signal (sub)

BUGS

PROGRAM
kwic - key word in context

USAGE
kwic [omit-filename]

FUNCTION
'kwic' is a program used to help make permuted indexes for manuals, such as the one at the beginning of the REGULUS manual. 'kwic' reads from its standard input, and writes to its standard output.

The 'omit-filename' is the name of the file that contains any words which you do not want 'kwic' to treat as a key word. For example you probably do not want 'kwic' to treat the words "to", "a," or "and" as key words. The omit-filename may have only one word per line.

The 'kwic' program will most commonly be found in a pipeline such as the following:

```
kwic omitwords < file | sort -d | unrot > INDEX
```

which will cause 'kwic' to read from the file 'file', writing to its standard output, without creating new lines for any words found in the file 'omitwords'. This command then sorts the output from 'kwic', and calls the 'unrot' program to center the output around the middle of the page.

A more complete description of how these programs interact and how to make a permuted index can be found in 'pindex (misc)'.

SEE ALSO
pindex(misc)
unrot(cmnd)
sort(cmnd)

BUGS

PROGRAM
ld - 68000 link editor

USAGE
ld [-nslXZUoIr] name ...

FUNCTION
'ld' combines several object programs into one; resolves external references; and searches libraries. In the simplest case the names of several object programs are given, and 'ld' combines them, producing an object module which can be executed on the 68000. The output of 'ld' is left on 'a.out' (or the file name specified by -o). This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important. Libraries are assumed to be standard ar (cmnd) format (either magic number 177555 or magic number 177545 library formats can be searched).

'ld' understands several flag arguments which are written preceded by a `-'. Except for -l, they should appear before the file names.

-n build a shared text executable module. Implies a special magic number.

-s `strip' the output, that is, remove the symbol table and relocation bits to save space.

-r put the relocation bits on the output file (default is no relocation bits on the output file).

-I don't output error messages for 16-bit address overflow.

-U take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-o interprets the argument immediately following it as the name of the output file. If this argument is not

specified, the name of the output file is a.out.

-l This option is an abbreviation for a library name. -l alone stands for `/lib/lib7.a', which is the standard system library for assembly language programs. -lx stands for `/lib/libx.a' where x is any character. A library is searched when its name is encountered, so the placement of a -l is significant.

-X Save local symbols except for those whose names begin with `L'. This option is used by 'ld' to discard internally generated labels while retaining symbols local to routines. If the -X flag is not specified, ld puts only global symbols into the symbol table.

-Znnnnnnnn Define nnnnnnnn as beginning hex address for text segment. This address defaults to 0, but can be specified as any hex number between 0 and 0FFFFFFFF hex. This option is especially useful for stand-alone programs. nnnnnnnn must be a hex number -- lower case a-f or upper case A-F are both allowed.

-Dnnnnnnnn Define nnnnnnnn as beginning hex address for data segment. This address defaults to next byte after end of text segment, but can be specified as any hex number between 0 and 0FFFFFFFF hex. This option is especially useful for stand-alone programs. nnnnnnnn must be a hex number -- lower case a-f or upper case A-F are both allowed.

-Bnnnnnnnn Define nnnnnnnn as beginning hex address for bss segment. This address defaults to next byte after end of data segment, but can be specified as any hex number between 0 and 0FFFFFFFF hex. This option is especially useful for stand-alone programs. nnnnnnnn must be a hex number -- lower case a-f or upper case A-F are both allowed.

FILES

/lib/lib7.a	standard library
/lib/lib6.a	version 6 i/o routines
/lib/libF.a	fast floating point library
/lib/libE.a	IEEE floating point library
a.out	output file

SEE ALSO

as(cmnd), ar(cmnd), cc(cmnd), float(misc), a.out(files)

DIAGNOSTICS

Error messages begin with a : and are meant to be self-explanatory.

PROGRAM
link - link a file to a new name

USAGE
link <existing name> <new name>

FUNCTION
link creates a link to the existing file and gives it the name of its second argument. If the second argument is a directory name, link will create a file with the same name as the original in that directory. The existing name must not be a directory.

FILES

DIAGNOSTICS
linking a file to itself is not overly useful and will produce an error.

SEE ALSO
ln (cmnd)
cp (cmnd)

BUGS

PROGRAM
list - ascii source lister

USAGE
list [-<param> -<param> ...] [file...] [-<param> -<param> ...] [file..

FUNCTION
'list' prints one or more files according to given specifications. The default specifications will print each line preceded by a line number, 60 lines per page, expanding tabs every 4 spaces, with a maximum line length of 85 characters. A default page header will be used. Output will be written to the standard output file.

Parameter arguments are a series of characters and numbers preceded by a hyphen ('-'). An hyphen alone as an argument specifies that the standard input file is to be read as an input. If no files are specified on the command line with the 'list' command, the standard input will automatically be input.

c CRT display mode. Default is false. A header and 20 lines are displayed to fill a 24 line screen. Before each page (including the first) the user is prompted by the message "(CR to go)".

f Fold output. The default is to not fold lines. The folded portion of the line is indented 4 spaces.

l# Set the page length to #. Default is 60 lines.

m# Set left margin to #. Default margin width is 4. Can not be set larger than 49 columns.

n Don't output line numbers.

p Direct output to the spooler. Default is to write output to standard output.

t# Set the tab value to #. Tabs will be set every # spaces. Default spacing is every 4 spaces.

u Lower case letters will be shifted to upper case. The default is to not do this.

w# Set page width to #. Default page width is 85 characters. Lines longer than the current width will be truncated unless folding is taking place.

:# Stop printing after page #. The default is to print all pages.

+# Start printing with this page #. The default is to start printing with page 1.

T write file named by the next argument transparently to the output. This is a useful way of sending device dependent escape sequences without printing a header or doing any tab expansion.

h take the next argument as a custom header specification. The header is displayed at the top of each page and can contain any characters. There are 4 format specifications which the list program will act on:

%f is replaced by the name of the current file.
%t is replaced by a 24 character repesentation of the modification time of the current file.
%u is replaced by the value of the LOGNAME environment variable.
%p is replaced by the page number in the format '%4d'.

The default header looks like:
"%f Page%p %u %t"

If the fold option is not in effect, lines which are longer than the current line length will be truncated at the right hand margin. Parameters are not reset between files.

FILES

SEE ALSO
cat (cmnd), more (cmnd)

BUGS

PROGRAM
ln - link a file to a new name

USAGE
ln <existing name> <new name>

FUNCTION
ln creates a link to the existing file and gives it the name of its second argument. If the second argument is a directory name, ln will create a file with the same name as the original in that directory. The existing name must not belong to a directory.

FILES

DIAGNOSTICS
linking a file to itself is not overly useful and will produce an error.

SEE ALSO
cp (cmnd)

BUGS

PROGRAM
lo68 - 68000 link editor

USAGE
lo68 [-nslXZUoIr] name ...

FUNCTION
''lo68 is identical to the 'ld' command and should be linked (see ln (cmnd)) to '/bin/ld' under REGULUS systems.

For more information on this command, see the documentation under 'ld' (cmnd).

SEE ALSO
ld (cmnd)
c68 (cmnd) which is a link to '/bin/cc'
as68 (cmnd) which is a link to '/bin/as'

PROGRAM
login - log into the REGULUS system

USAGE
login [username]

FUNCTION
'login' is used to gain access to the REGULUS system.

If there is no argument, 'login' will prompt for a username. After 'login' has a username (either from the command line or by prompting), it will prompt for a password. Echoing is turned off during the typing of the password.

'login' will verify that the username exists as a valid REGULUS user and that the password matches the stored password.

If there is a match, 'login' will change into the directory and execute the program specified for this user in the password file. 'login' will also update the current user list to show the new user and sets the users initial environment to contain the variables LOGNAME, PATH and HOME.

If the username or password do not match, 'login' will prompt again.

FILES
/etc/passwd password file with user information.
/etc/utmp list of current users

DIAGNOSTICS
'Login incorrect' for invalid usernames or passwords

NOTES
Must be set-uid-root.

PROGRAM
logname - print login name

USAGE
logname

FUNCTION
'logname' prints the current user's login name. The environment variable "LOGNAME" is used to determine the current user's login name.

FILES

SEE ALSO
env (cmnd)
login (cmnd)
who (cmnd)
whoami (cmnd)

BUGS

PROGRAM
lpr - line printer spooler

USAGE
<commands> | lpr
lpr [filename...]

FUNCTION
lpr is the REGULUS line printer spooler. lpr takes it input from the standard input, and spools it in the spooling directory, for printing as soon as the line printer is available. 'lpr' also accepts one or more filenames on the command line as input.

FILES
/usr/spool/lpr
spooling directory
/usr/spool/lpr/cfaXXXXX
spooled file
(XXXXX is a unique number formed from pid)
/usr/spool/lpr/tfaXXXXX
temporary file - becomes dfaXXXXX
/usr/spool/lpr/dfaXXXXX
holds pertinent data about the spooled file
also used by lpd(maint)

DIAGNOSTICS

SEE ALSO
lpd(maint)

BUGS

PROGRAM
ls - list information on a file or directory

USAGE
ls [-adgilrstux] [file1] ...

FUNCTION
'ls' will list information about its file arguments to its standard output according to the options on the command line. With no arguments, 'ls' lists the contents of the current working directory alphabetically, one name per line.

If a filename is given as an argument, and the file is not a directory, 'ls' will list its name along with any requested information. If the argument is a directory, the contents of the directory are listed.

The following flags are available:

a list even those files whose first letter is a '.'

d treat a directory as an ordinary file in the listing.

g print the group name instead of the user name in a long listing

i print the index number as the first field

l give a long listing of the file (see later text)

p precede each filename with its pathname

r reverse order of sort

s list size of file in 512 byte blocks rather than bytes

t sort on date modified, most recently modified first

u sort on date accessed, most recently accessed first

x display date accessed instead of date modified in long listing

The long listing has the following format:

The first field is 10 characters wide and shows the type and permissions associated with the file or directory. The first character shows the type of a file.

d file is a directory

c file is a character special file (device)

b file is a block special file (device)

p file is a fifo (named pipe) file

- ordinary file

The next three characters list the read, write and execute permissions (in that order) for the owner of the file.

r owner has read access

w owner has write access

The 'x', 's' and 'S' characters share the same position in the listing.

x if an ordinary file, the owner may execute (run) the file by typing the name of the file. If the file is a directory, the owner has search permission into the directory.

s set-user-id on execution (see 'setuid (sys)').

S set-user-id on execution bit is on, but the execute bit is not on.

- the permission described at this position is being denied.

For instance, an 'r-x' field means that the owner has read and execute permission, but no write permission.

The second set of 3 characters are the permissions for the group. These are interpreted the same as above with the following changes:

s set-group-id on execution (see 'setgid (sys)').

S set-group-id on execution, but group execute permission is not allowed.

The last 3 characters are for permissions for others who are not in the owners group. These are interpreted as for the owner with the following change:

t if the file is a directory, restrict the removal of files to the owner of the file (or the superuser). If the 't' bit is not on, only write permission on the directory is required to remove a file.

If the file is an execution module produced by the loader, save the text portion of the file in the swap space after the file has been executed. This

may speed up the initial load time of some programs. Note however that setting this bit on non-directory files is restricted to the superuser.

The second field is the name of the owner or group of the file. If the user-id or group-id do not match a name, the numerical id is given.

The third field specifies the number of links that exist to the file. Each unique name to a file uses one link.

The fourth field is the date the file was last accessed or modified.

The fifth field is the size of the file. Unless blocks are specified, the size is given in bytes. If the file is a character or block special file (device), the major and minor device number is given in this position.

The last field is the file name.

FILES

/etc/passwd	gets name of file owners from here
/etc/group	gets name of file groups from here

DIAGNOSTICS

SEE ALSO

chmod (cmnd), dir (cmnd), ln (cmnd), setuid (sys), setgid (sys), mknod (cmnd)

BUGS

PROGRAM

mail - send and receive mail

USAGE

```
mail person ...
mail [-pd] [-f file]
```

FUNCTION

'mail' allows the user to send and receive mail. If there are any non-flag arguments, mail sends its standard input to the persons specified by the arguments. The persons can either be login names (found in /etc/passwd) or they can be file names. Mail sent to login names is appended to that person's mailbox, a file in the directory "/usr/spool/mail". Mail sent to a file name is appended to that file.

Sending of mail can be aborted by entering a line beginning with a tilde. The line with the tilde may contain the name of a file in which to save the unsent mail. If no name is specified the unsent mail is saved in the file "dead.letter".

If only flag arguments are specified, mail displays messages from the user's mailbox, prompting for a command after each message. When displaying messages which are longer than one screen, mail pauses after each screen and prompts "CR to continue... ". Typing just a carriage return causes the next screen to be displayed; typing anything else followed by a carriage return stops the display.

Mail accepts one of the following commands after each message is displayed:

CR	Go on to the next message. If "auto-delete" mode is selected (see flags) the message is deleted, otherwise it remains in the mailbox.
d	Delete this message and go on to the next message.
p	Print the same message again.
m [persons]	Mail this message to the named persons, yourself is the default.
s [files]	Save this message in the named files, "mbox" is the default.
x	Exit without changing mailbox
q	Quit, copying unexamined mail
?	Print a command summary

The following flags allow the user to specify alternate modes of reading mail:

-f The next argument is the name of the file to be used as a mailbox instead of the user's default mailbox in "/usr/spool/mail".

-p All of the messages are printed without pausing for commands or after each screen. The mailbox is not changed.

-d Each message is deleted automatically after being displayed. This is refered to as "auto-delete" mode. A message can be "saved" in this mode by using the "m" command.

FILES

/usr/spool/mail/* - default mail boxes
dead.letter - default file in which to save default mail
mbox - default file to store messages saved with 's' command

DIAGNOSTICS

SEE ALSO

BUGS

PROGRAM

man - print manual entries

USAGE

man command [class]

FUNCTION

'man' displays the documentation for the specified item if it exists. Documentation is available for each of the sections from the REGULUS manual. Each section is searched in order, and the first occurrence of the item is then shown.

To specify a particular manual section, use the 'class' option. This is used where commands or subroutines have the same name. The classes are: cmnd, sys, subs, files, stdio, v6, v7, sys3, dev, misc and maint. 'v6', 'v7', and 'sys3' are the documents for the compatibility subroutines and system calls.

The main difference between help(cmnd) and man(cmnd) is that man displays the manual entry in a format suitable for printing while help displays the documentation in a smaller amount of space fit more for a terminal.

FILES

/doc/man/[class]	
/doc/man/[class]/macro	Section specific macros
/doc/man/mdef	Printing document macros

SEE ALSO

help (cmnd)

BUGS

PROGRAM
mc - multi-column filter

USAGE
mc [-c#] [-l#] [file ...]

FUNCTION
'mc' is a filter designed to save space by printing short lines in a number of columns. The arguments are:

-c# Number of columns. If not given, 5 is used. There are no spaces between the 'c' and the number.

-l# Line length. If not given, 78 is used.

The remaining arguments are assumed to be files. Each file is opened in turn and printed. If there are no file arguments, 'mc' reads it's standard input.

The size of the columns is determined by dividing the line length by the number of columns. Lines which are longer than the size of a column are truncated. There is a maximum line length of 132 characters. Unreasonable combinations of number of columns and line lengths are ignored. In these cases 'mc' reverts to using the default values.

FILES

DIAGNOSTICS
Unknown flag arguments will be complained about.

SEE ALSO

BUGS

PROGRAM
mesg - turn messages on or off

USAGE
mesg [y] [n]

FUNCTION
'mesg' allows the user to restrict the ability of other users to write on the user's terminal. Without any arguments, 'mesg' toggles message reception. The previous status of messages will be displayed.

FILES
/dev/tty* - user's terminal device

SEE ALSO
write (cmnd)

PROGRAM
mkdir - make a directory entry.

USAGE
mkdir dir [dir ...]

FUNCTION
'mkdir' makes empty directories of the names given as arguments. The directories are created with access permissions of 0777 subject to the current value of the umask. See 'chmod (cmnd)' for more on access permissions. The entries '.' and '..' are made in that directory automatically.

Ownership of the '.' directory will be the same as the person who makes the directory. The directory '..' will belong to the owner of the parent directory.

The user must have write permission in the parent directory.

FILES

DIAGNOSTICS
'mkdir' will have an exit status of zero if all went well, otherwise it will have a terminating status of -1 and will issue an error message.

SEE ALSO
rmdir (cmnd)

NOTES
Must be set-uid-root.

PROGRAM
mkver - make a SCCS version file for a C program

USAGE
mkver [-e] [-v] [-f filename] ["optional string"]

FUNCTION
This command is used to make a version file of the form

```
char *compiled = "@(#) Tue Jul 26 13:15 1983";
```

that can be used by 'what' (cmnd) to determine when a program was compiled. By default, 'mkver' will output a string in the above format with the current date to a file called 'version.c'. Then, version.c is typically compiled and linked (using ld (cmnd)) with the other '.c' files of a program.

If the '-e' option is used, the version string will be output to the standard output in addition to the 'version.c' file.

The '-v' option causes 'mkver' to name the C character array 'version' rather than 'compiled' ('compiled' is the default as in the above example).

You may specify the newly created filename to be something other than 'version.c' by using the '-f filename' option.

If you include a string as an argument to 'mkver' (enclosed in quotes to pass it as one argument), it will be included in the string it creates.

This command is most commonly used in 'buildfiles' (see build (cmnd)) in the form

```
mkver -e -v "Version of";cc -L mkver.c version.c
```

FILES
version.c is the default output file created

SEE ALSO
build (cmnd)
what (cmnd)

BUGS

PROGRAM
more - print files a page at a time

USAGE
more [file...]

FUNCTION
'more' prints one or more files on the standard output device a page at a time. If no arguments are given 'more' reads from the standard input.

At the prompt '< more >' or '< next file >' the following commands are allowed :

space	continue to next page.
'n'	go to the next file.
'b'	go back to the beginning of the current file.
'f'	print current file's name and current line number.
'q'	quit.

A space at the '< next file >' prompt will give you the next file. If more than one file is specified each file will be preceded by a three line header consisting of two lines of asterisks separated by the file's name.

If the output is redirected the file or files will be concatenated together with headers if needed.

SEE ALSO
cat (cmnd), list (cmnd)

BUGS

PROGRAM

mv - move files or directories

USAGE

```
mv [-f] [-i] src_file dest_file
mv [-f] [-i] file... directory
mv [-f] [-i] olddirname newdirname
```

FUNCTION

'mv' either renames files or directories, or moves a file to another location. Multiple files may be moved if the destination file is a directory. The '-i' flag will ask the user before removing any file which 'mv' would overwrite. The '-f' flag will attempt to remove any file which 'mv' would overwrite regardless of its permissions without notifying the user.

When changing the name of a directory, the new directory name must be placed in the same parent directory as the original directory. It is not legal to rename a directory to an existing directory name. Moving a directory to an existing file will overwrite the file.

SEE ALSO

cp(cmnd)
mvdir(maint)

NOTES

If the target file or directory is on a different file system from the source file, the file will be copied to the destination and removed from the current location. Subsequently the file will be owned by the copier who is not necessarily the original owner.

Must be set-uid-root.

PROGRAM
newgrp - change group id

USAGE
newgrp group

FUNCTION
'newgrp' allows one to change their group id by specifying the new group name and then typing the appropriate password if requested. If the password is correct a new shell is executed with the group ID set to that of the specified group.

A password is requested if the group has a password and the user does not, or if the group has a password and the user is not listed as a member of the group. The super-user does not need to specify a password in order to change groups.

If 'newgrp' is called with no arguments, it will change your group to your original login-group.

FILES
/etc/password
/etc/group

SEE ALSO
login (cmnd)
su (cmnd)
newgrp (cmnd)

BUGS
There is not a convenient way to enter a password into /etc/group.

PROGRAM
news - display the latest news

USAGE
news

FUNCTION
'news' is a program used to keep you informed of current events.

FILES
/etc/news - the news file

SEE ALSO

BUGS

PROGRAM
nice - adjust a processes priority on execution

USAGE
nice [+/-#] [-h] [-l] [-r] command [arguments]

FUNCTION
'nice' causes 'command' to be executed with the specified priorities. Without any flags the command will be executed at the lowest priority. The lowest priority value is zero, and the highest is 50.

+#	increase command priority.
-#	decrease command priority.
-h	execute the command at the highest priority.
-l	execute the command at the lowest priority.
-r	execute the command as a real time process.

The highest priority available is 50. The lowest priority is 0.

SEE ALSO
nice (sys)

PROGRAM
nm - print symbol table

USAGE
nm [-egpqxdtba] [file...]

FUNCTION
'nm' prints the symbol table from the output file of a 68000 assembler (as68) or loader (lo68) run. If the specified file is an archive, nm will be run on each of the individual object files included. If no options are specified all symbols will be displayed. Otherwise, any option selected will cause all symbols of that type to be displayed. Each symbol name is followed by its value and one or more of the following type descriptors (the relevant option is prefixed).

e equ (equated)
g global
q equreg (equated register)
x external
d data
t text
b bss
a abs (absolute)

The output is not sorted and thus appears in the order that the symbols appear in the file's symbol table.

FILES
a.out default file name

SEE ALSO
ld (cmnd)
cc (cmnd)

BUGS

PROGRAM
nohup - run command immune to hangups, quits

USAGE
nohup command [arguments]

FUNCTION
Nohup executes command with hangups and quits ignored. This is not a seperate command, but is a builtin command of the shell.

SEE ALSO
sh(cmnd), nice(cmnd), signal(sys)

PROGRAM
od - octal dump

USAGE
od [-bcdhox] [file] [[+]offset[.b]]

FUNCTION
'od' prints the specified file in one or more formats. The default format is to print in octal bytes. 'od' prints 8 integers or 16 bytes to a line, depending on the format chosen. One or more formats may be used together, which will result in a line printed for each format.

The flags have the following meaning:

b	octal bytes
c	ascii characters
d	decimal integers
h	hexidecimal integers
x	hexidecimal integers
o	octal integers

A starting offset may be given as the last parameter. If none is given, 'od' will start at the beginning of the file. The offset will be interpreted as blocks of 512 bytes if a '.b' is added. If the offset is preceded by a zero 'od' will print the addresses in hexidecimal. If the file argument is omitted and an offset is still desired it must be preceded by a plus '+'.

DIAGNOSTICS
An exit status of 0 is returned if the file could not be opened, and exist status of -1 is returned if it is presented with bad options.

SEE ALSO
strings (cmnd)

BUGS

PROGRAM

passwd - change password

USAGE

passwd [user]

FUNCTION

'passwd' allows a user to change his password. The user can be specified by name or user id number.

'passwd' prompts for the old password if the password for the current user is being changed. The old password must be entered correctly before 'passwd' will accept a new password. 'passwd' disables echo so that passwords are not displayed as they are being entered.

The new password must be six characters long, four characters if it contains special characters or mixed upper and lower case characters. Shorter passwords can be entered if the user is persistent.

After a password has been accepted the user is asked to enter it again. If this second attempt does not match the first attempt the password is not changed.

Super-user can change the password of any user.

FILES

/etc/passwd password file

DIAGNOSTICS

sorry	not super user or incorrect old password
can't open file	the specified file could not be opened
try again later	the password file is being updated by someone e
mismatch	the second attempt did not match the first

SEE ALSO

login (cmnd)
passwd (files)
cryptpw (subs)

BUGS

PROGRAM
ps - process status

USAGE
ps [-adexfl]

FUNCTION
'ps' lists process information according to the flags specified. 'ps' reports the following information:
the process flags -- in octal and cumulative
01 in memory
02 real time task
04 task is locked in/out of memory
10 task is being swapped
20 task is being traced by another task
the status (Z=zombie, T=stopped, R=running)
the owner (user-id)
the process id
the parent task's process id
the scheduling priority
the priority of the process
the size of the process in K (1024) bytes
the controlling tty
the cumulative execution time
and the maketask arguments or identity of the process.

Four options (a, d, e, x) control which processes are reported upon. If no option, then report only on the processes in the current process group (i.e. connected to this tty).
-a lists all processes except group leaders and processes without an associated terminal
-d lists all processes except group leaders
-e lists all processes
-x lists all processes with the same user-id (owner)

Two options (f, l) control what information is reported about each process. If no option, display only the process id, associated tty, execution time, and command line.
-f output a Full listing. In addition to the default listing content, output the owner, parent process id, execution priority, and priority.
-l produce a Long listing. In addition to the default and full listing content, output flags and status information.

FILES

/dev/mem	for process information.
/tmp/.psttyinfo	this is a file built by 'ps' if it doesn't exist. It contains a table relating special devic terminal names. It should be deleted any time ter added (using mknod) or deleted.

SEE ALSO
kill (cmnd)

BUGS

The Regulus kernel tables examined by 'ps' can change during the execution of the command; this can produce puzzling results.

Note that 'ps' searches through the user stack to figure out the process arguments. Hence, if a user has damaged his stack, 'ps' will not do a good job of reporting the command line.

PROGRAM
pwd - print current working directory

USAGE
pwd

FUNCTION
'pwd' will print the full pathname of the current working directory.

FILES

DIAGNOSTICS

SEE ALSO
cd (cmnd)
wd (cmnd)

BUGS
This command is a built-in function of the shell.

PROGRAM
rm - remove files

USAGE
rm [-i] [-f] [-r] [-v] name [name2 ...]

FUNCTION
'rm' is used to remove files from the REGULUS filesystem. The names given as arguments may be either file or directory names. If the file is write-protected, the user will be prompted with the filename and the permission flags for the file. An answer of 'y' will remove the file. Any other response will skip that file. The options of 'rm' are:

-i interactive remove. This flag must be specified if directory names are given. The user is prompted with each filename before it is acted on by remove. If the user responds with a 'y', the file is deleted. If he responds with a 'x', 'rm' will exit immediately. Any other response will skip that file.

If a directory name is given, the contents of that directory are examined and the user is prompted for each file.

-f force remove. The user is not notified of errors or failure to find given files. In addition, protection bits on the files are not examined.

-r recursive remove. For each directory specified, that directory is recursively searched for files to delete. If all the files, including subdirectories, are deleted from a directory then "." and ".." are deleted and the directory name deleted. Note that this is a very dangerous option.

-v verbose mode. This prints out the partial path name of every deleted file. This is very handy for use with the "-r" option.

FILES

DIAGNOSTICS

SEE ALSO
rmdir (cmnd)
dsw (cmnd)

BUGS

PROGRAM
rmdir - remove a directory entry.

USAGE
rmdir dir [dir2 ...]

FUNCTION
'rmdir' removes a directory from the filesystem provided that the following conditions are met:

1) there are no files still residing in the directory (except '.' and '..')
2) the user has write permission for the directory specified
3) the user has write permission for the parent directory

If the parent directory is sticky, the user may only remove a file if he is in fact the file's owner as well as having the appropriate permissions.

FILES

DIAGNOSTICS
'rmdir' will have an exit status of zero if all went well. Otherwise an error message will be issued.

SEE ALSO
mkdir (cmnd), rm (cmnd)

NOTES
Must be set-uid-root.

PROGRAM
roff - text formatter

USAGE
roff [-t] [-s#] [-o#-#] [-#] [+#] [file...]

FUNCTION
'roff' accepts one or more files as input, formats them based on various commands and then directs the formatted output to the standard output. The following command line flags are understood:

-t the page length and width is set up for a terminal page.

-s stop output at the end of each page, and continue upon receiving a carriage return. This is for printing on non-continuous form paper.

-o specify a range of pages to format. The first number is the page to start on and the second is the page to terminate on. If one of the numbers is ommitted then roff will start or terminate on the first or last page respectivly.

-# another way to specify the last page to format.

+# another way to specify the first page to format.

Boolean Variables

Several 'roff' commands are actually boolean variables which may be turned on or off.

.fi - turn filling on <default>
.nf - turn filling off
.ad - turn line adjusting on <default>
.na - turn line adjusting off
.ar - arabic page numbers <default>
.ro - roman page numbers
.tx - build a table of contents

Line filling means that 'roff' will read as many lines of text from the file as are reguired to fill a line to the current line length less the size of the current indent. If adjusting is also enabled, after the line is filled 'roff' will pad the line with spaces until the line is exactly as long as the line length. This has the effect of producing perfect left and right margins.

The format for page numbers is specified as either arabic or roman numbers.

The words following '.tx' are put into a file called

'contents' followed by the page number that that particular '.tx' command occurred in.

File Set-up

The next set of 'roff' commands are used for setting up the format of the document in general, although they can be adjusted at any place in the document.

.ds - double spacing
.ss - single spacing <default>
.ls - line spacing
.ll - line length <default 70>
.pl - page length <default 66>
.po - page offset <default 15>

Line spacing, line length and page length require a numeric argument to define the appropriate number of spaces between lines, the number of characters on a line, and the number of lines on a page. 'ds' is equivalent to 'ls' with a parameter of '2' and similarly 'ss' is 'ls' with an argument of '1'.

The page offset is the number of character spaces added to every line before the line is printed (i.e. the left margin). The page offset and line length affects every line in the printed document.

Page Format

The next set of 'roff' commands deal with the format of page headers and page footers.

.he - page header
.fo - page footer
.eh - even page header
.ef - even page footer
.oh - odd page header
.of - odd page footer
.m1 - margin 1 <default 2>
.m2 - margin 2 <default 2>
.m3 - margin 3 <default 2>
.m4 - margin 4 <default 2>

Headers and footers are used to make 3-part titles and require as an argument the header (or footer) string. The titles are separated by a single character that does not appear in the title. The parts of the title are printed at the left edge, center and right edge of the page. A percent sign '%' appearing in any of the titles will be replaced by the current page number. As an example, the REGULUS documentation uses the following footer line:

The single quote character "'" is used to separate the title strings.

Margin 1 is the number of lines between the top of the page and the header. Margin 2 is the number of lines between the header and the text of the page. Margin 3 is the number of lines between the text and the footer and margin 4 is the number of lines between the footer and the bottom of the page.

Indention

The next set of 'roff' commands are used to control indention.

.in - indent text
.ix - indent at the next new line
.ti - temporarily indent text

These commands take a signed numeric parameter to describe the direction and number of spaces which you wish to indent. Indent causes a break and starts indention with the next line. Temporary indent also causes a break and will then adjust the left margin for the next line of input text. The 'ix' command does not cause an immediate break, but indention will occur at the next new line. Negative indentions decrease the left margin and positive indentions increase the left margin.

Pagination

The next set of 'roff' commands control pagination.

.bp - begin new page
.sk - skip page
.ne - conditional pagination
.pa - paginate

The 'bp' and 'pa' commands force a new page immediately. If they are given a numeric argument the new page will be numbered explicitly, otherwise the page will be one greater than the current page. The skip page command 'sk' takes a numeric argument and leaves one or more empty pages (paginate will act the same way if given an argument). The conditional paginate command is used to request 'n' contiguous lines on the current page. If the requested number of lines will not fit on the current page 'roff' will skip to the top of a new page.

Lines

Line formatting and skipping are controlled by the following commands.

.sp - space (empty line)
.ce - center line
.bl - blank line

'sp' skips 1 or more lines, unless the current line is at the top of a page. In this case, no lines are skipped. 'bl' will always skip 1 or more lines, even at the top of the page. Lines will be skipped until the number of lines specified or the bottom of the page, whichever comes first.

The centering command will center the line immediately following. If a parameter is given to 'ce' it will center the next specified number of lines.

General Commands

There are a number of general commands which might appear anywhere in the document.

.ta - set/reset tab stops
.ig - ignore the next line
.br - break
.li - literal text
.cc - control character <default = '.'>
.so - switch source file
.nx - next file
.de - define macro

'roff' replaces leading tabs in the input stream with spaces. The current default number of spaces per tab is 4. If a parameter greater than zero is given, it becomes the new tab size. A zero will return the tab stops to the default.

A break causes 'roff' to stop filling the current line and to output it. Lines that begin with a space or a tab will also force a break.

Literal causes the next specified number of lines to be output exactly as they appear in the input stream.

The control character command allows the user to alter the character which normally introduces 'roff' commands and macros. This character is normally a period '.'.

The switch source file command causes 'roff' to read the named file into the input stream at the place where the command is found. After the file is read, 'roff' continues with the current file.

The next file command causes 'roff' to start formatting another file as if it was an extension of the current one. The original file is not returned to.

Define macro takes a macro name as a parameter. User defined macros may not currently define their own parameters. All successive lines up to one which starts with two periods '..' will be considered part of the macro. User macros are used in the same way as 'roff' built-in commands.

Command Format

All 'roff' commands must begin in column one. Every command is of the form:

control character, command name, space, optional parameter

Text which is not directly related to the macro command on the line with a macro will either be ignored or interpreted as part of the command.

Miscellaneous

Sometimes it is desirable to have a line begin with a period (.) which is usually the control character, causing 'roff' to read the rest of the line as a command. In order to have a text line begin with a period, you can use an 'invisible space' immediately preceding the period. A backslash immediately followed by an ampersand is interpreted by 'roff' as an 'invisible space'.

Roff Command Summary

.ad - turn line adjusting on <default>
.ar - arabic page numbers <default>
.bl - blank line
.bp - begin new page
.br - break
.ce - center line
.cc - control character <default = '.'>
.ds - double spacing
.ef - even page footer
.eh - even page header
.fi - turn filling on <default>
.fo - page footer
.he - page header
.ig - ignore the next line
.in - indent text
.ix - indent at the next new line
.li - literal text
.ll - line length
.ls - line spacing <default = 1>
.m1 - margin 1 <default = 2>
.m2 - margin 2 <default = 2>
.m3 - margin 3 <default = 1>
.m4 - margin 4 <default = 3>
.na - turn line adjusting off
.ne - conditional pagination
.nf - turn filling off
.nx - next file
.of - odd page footer
.oh - odd page header
.pa - paginate, like .bp
.pl - page length
.po - page offset <default = 0>
.ro - roman page numbers
.sk - skip page
.so - switch source file
.sp - space (empty line)
.ss - single spacing <default>
.ta - set/reset tab stop
.ti - temporarily indent text
.tx - build a table of contents

PROGRAM
run - execute a command

USAGE
run [-priority] [-roc] command [args]

FUNCTION
'run' executes a command.

-#	command task startup priority.
-r	real time task.
-o	overlayed task.
-c	copied task, not compatible with the '-o' flag.

SEE ALSO
maketask(sys)

PROGRAM

sendc68 - 68000 downloader

USAGE

sendc68 [-r] [-d delay] [-s start] [-e end] [-] objectfile [ttyname]

FUNCTION

'sendc68' reads the objectfile, which must be in the a.out format produced by the 68000 linking loader lo68. Sendc68 converts this executable object code into the absolute ASCII load format (S records) acceptable by the MACSBUG monitor and then sends these S records to the 68000 via a serial line.

Locations 400 hexadecimal through 1000 hexidecimal are MACSBUG ram and will not be loaded. In order to override this, the '-s' and '-e' flags can be used to alter the addresses which are not to be loaded. With 'start' being the first address to not load, and 'end' being the last address. By setting the ending address to be smaller than the starting address all locations will be considered legitimate.

The optional argument '-r' will only output a newline, rather than the standard newline-carriage return. This is for use by systems which translate newline to carriage return-newline.

The '-d delay' flag allows the user to specify a line delay. This is for slowing the output of sendc68 when outputting to a slow device.

If the optional final argument 'ttyname' is present, this is the name of the special file (ie. /dev/tty?) to use to download the 68000. If this optional third argument is not present, the standard output file is used.

If the optional "-" argument is specified, sendc68 does not clear the bss segment during the load. This results in a faster load, but all bss memory will have random initial values.

SEE ALSO

c68 (cmnd), as68 (cmnd), lo68 (cmnd), a.out (files)

PROGRAM

setstack - set the stacksize

USAGE

setstack file stacksize [breaksize]

FUNCTION

Sets the stack entry or break size in an executable file to the given values. The values are rounded up to the next 1K value. If the stacksize is 0, the entry in the executable file is not changed. If the breaksize is not given, the entry in the executable file is not changed. The entries are only advisory on a system with a Memory Management Unit.

SEE ALSO

size (cmnd)
a.out (file)

BUGS

PROGRAM
sh - REGULUS shell (command line interpreter)

USAGE
sh [-cefinpstvVxX] [arg ...]

FUNCTION
'sh' is the standard shell (or command line interpreter) for REGULUS. It contains many of the same features as the UNIX shell along with a number of enhancements that give greater flexibility and functionality to the user. This document attempts to describe both the features and uses of 'sh'. It is designed to be both a learning tool for the beginner and a reference manual for the advanced user.

Certain concepts basic to using the shell are first discussed to acquaint the new user to the REGULUS method of command interpretation. The various substitutions on the command line are then discussed in the order that the shell applies them.

Shell arguments

The shell accepts arguments on the command line just as any other REGULUS program may. The following arguments have special meaning to the shell:

none If there are no arguments and the first character of the shells invocation name (i.e., **argv) is the character '#', the shell assumes it is a login shell. If the shells input file is a terminal, the shell will be interactive and prompt for it's input.

-c A command list (see below) is read from the single next argument which must be present. This is commonly the way that '!' commands are run from programs.

-e Exit on abnormal command exit. If any command that the shell runs exits with a non-zero status, the shell will immediatly exit with that status. This is most often used with command files or the '-t' or '-c' arguments to prevent the shell from executing a multiple command line when one of the commands fails.

-f Fast start. No automatic sourcing of '~/.nshrc' or '~/.login' is done. This causes the shell to start much faster.

-i The shell is to be interactive and prompt even if it appears that the input is not from a terminal.

-n Commands are parsed but are not executed. Often used to help debug command files.

-p No device '/dev/tty' available for programs. The shell

normally attempts to open '/dev/tty' as the input or output to prevent programs from corrupting the shell's own inputs. If this device is not available (for instance, from 'init (maint)' in running the '/etc/rc (files)' command file), this flag informs the shell and prevents a spurious error message.

-s Command input is taken from the standard input.

This flag is not effective with the '-c' option.

-t A single line of input is read and executed as a command list. Multiple lines may actually be read if the newline that ends each line is escaped with a backslant '\'. However, at most 256 characters will be read.

Only one of either '-c' and '-t' are allowed.

-v The 'verbose' shell variable (see below) is set after the shell reads the input source files '~/.login' and '~/.nshrc' (see below).

-V The 'verbose' shell variable is set before reading the input source files.

-x The shell variable 'echo' is set after the shell reads the input source files '~/.login' and '~/.nshrc' (see below).

-X The 'echo' shell variable is set before reading the input source files.

The first other argument is assumed to be the name of a command file (also known as a shell script file) which is to be read and executed, even if the execute permission bits are not on. All other remaining arguments are placed into the special shell variables named '1', '2', etc. up to '9'. If there are more than 9 arguments remaining, the 'shift' command (see below) must be used to access them. See 'Command files' below for more on arguments.

Special characters

The shell has a number of characters that are treated in a special way. The user is advised to remember these characters and use them with care. Each of these characters loses it's special meaning if it is preceeded by a backslant '\' or if it appears inside a pair of matched single quote characters ('). They may or may not be interpreted inside of double quotes (see the text for each command to be sure). The specific action of each of these characters is discussed below, but for reference the list of all shell special characters is given here.

These characters may be treated specially by the shell unless escaped:

( ) ! $ % & * ~ [ ] { } ; : # ' | \ < > ? / @

Command execution

Commands in REGULUS are simply a sequence of 1 or more words possibly separated by special characters and terminated by a carriage return (or newline). The command itself is normally the first word in the list, the other words being arguments to the command. A command and it's arguments are known as a 'command list'.

2 or more command lists may appear on a single line of input if they are separated by one of the following characters:

; the semi-colon is the simplest command separator. It functions just like a newline. The first command list is executed and then the 2nd command list, etc.

| the vertical bar introduces a pipeline. A pipeline forms a connection between commands so that the output of the first command may be used as input to the second command. Pipelines may be formed to connect a number of commands, the output of one connected to the input of the next, in a long chain.

Pipelines may be used in place of temporary files in the REGULUS environment. As an example, suppose a user wished to sort the contents of a file and have the sorted list sent to the printer. Using temporary files, the command would be like this:

```
sort file -o temp (the -o means output file)
lpr temp          (lpr prints the file)
rm temp           (rm removes the temporary)
```

Using a pipeline, the command would be:

```
sort file | lpr
```

() the parenthesis is used to execute a command or sequence of commands in a different environment than the current one. The commands enclosed within the parenthesis form a command list and are executed in a sub-instance of the shell, and will not change things in the current shell. This is most often used to prevent a 'chdir' command from affecting the current shell.

As an example, a user might say:

```
( cd somedir ; pwd ; ls )
```

to print a listing of 'somedir' without changing the current directory.

Background tasks

Normally, the shell waits for commands to finish before prompting for a new command. It is possible to have the shell not wait for a command to finish by ending the command list with the character '&' (ampersand). The command will be put into execution and the shell will immediately return and prompt for the next command.

The process id of the background command is printed by the shell so that the user may terminate or wait on the task. (see 'kill (cmnd)' or 'wait (cmnd)' for more information). Tasks in the background are started ignoring interrupts and hangups. This protects the background task from being accidentally killed by the user.

If a command that was started in the background terminates due to a signal, either generated by the user or from an internal error, the shell will print the process id and the reason for terminating.

I/O redirection

REGULUS supports a feature known as I/O redirection. This means that I/O may be re-directed independent of the workings of a given program, to suit the needs of the user. Many programs are written for REGULUS that will read their input from a terminal and write their output to the terminal. With I/O redirection, either of these may be changed.

REGULUS normally supplies each running command with 2 separate output streams, known as the 'standard output' and the 'standard error', and an input stream known as the 'standard input'. Initially, both of the output streams write to the users terminal and the input stream reads from the users keyboard. With I/O redirection, it is possible to separate these streams into different files. I/O redirection is used in the following way:

> name
>! name
Create the file 'name' and cause the standard output of the command to be placed in it. The second form causes the checks from the noclobber variable to be ignored (see 'noclobber' below). The command does not need to be told of this change as long as it normally writes it's output to the standard output (as most commands do).

>> name
>>! name

Open the file 'name' and cause the output of the command to be placed in it starting at the end of the file (appending). The second form causes the checks from the noclobber variable to be ignored (see 'noclobber' below).

Note that these redirections affect only the standard output stream, and that the standard error stream will still appear on the terminal.

>& name
>&! name
Create the file 'name' and cause the error output of the command to be placed in it. The second form causes the checks from the noclobber variable to be ignored (see 'noclobber' below). This form affects only the standard error output, not the normal output.

>>& name
>>&! name
As you might guess, '>>&' appends the error output like '>>'. The second form causes the checks from the noclobber variable to be ignored (see 'noclobber' below). Note that it also affects only the standard error stream, and not the standard output stream.

< name
!< name
Open the file 'name' and use the contents as the input for the command being executed. The second form causes the checks from the noclobber variable to be ignored (see 'noclobber' below).

As an example, the 'sort' command will read from the terminal and write to the terminal if given no other files. Using redirection, a user can take advantage of this feature in the following way:

```
sort < infile > outfile
```

This command will sort the contents of 'infile' and leave the sorted output in the file 'outfile'.

I/O redirection can be used in pipelines to affect the origin and destination of data.

```
sort < infile | uniq > outfile
```

In this example, 'sort' reads from the file 'infile', sorts the items and passes the sorted items along the pipeline to 'uniq', which removes duplicate entries and leaves the output in the file 'outfile'.

At the present time it is not possible to redirect the standard error output of separate commands inside a

pipeline. All of the error outputs of commands in a pipeline are directed together.

Variables

The shell has a number of 'built-in' variables that are available to the user at the command level. In addition, the user may create his own variables to use as desired. Variables may contain words or part of words, commands, filenames, directory names, or any other construction. In addition, variables may be used as toggles or switches. In this case, the shell is not concerned with the value of the variable but whether it is 'set' or 'unset'.

Variable names may be any number of characters in length and may contain only letters or numbers. They must begin with a letter. Both upper-case and lower-case letters are allowed and are treated as different letters by the shell. This means that a variable named 'lib' is different from 'Lib' and 'LIB'.

Built-in variables

The following is a list of the built-in variables available to the user. These variables are set up by the shell for every user. In addition, the shell refers to these in many cases. The values of these variables may be changed by the user but they may not be cleared completely.

home

The home variable is the full path name to the current users main directory. The default value is read from the environment if possible, otherwise it is set to the value as it appears in the password file (see 'passwd (files)').

Changing the value of this variable will also change the HOME environment variable.

path

The path variable is a list of path names of directories, separated by colons ':', spaces or tabs. The shell uses the values in this variable as the directories to search when looking for commands to execute. The current directory is always searched and need not be mentioned. The default path is read from the environment if possible, otherwise it is '/bin:/usr/bin' for normal users and '/bin:/usr/bin:/etc' for the superuser.

Changing the value of this variable will also change the value of the PATH environment variable.

child

The child variable contains the process id's of all of

the current shells children (i.e., background tasks). This variable can be used to determine which tasks to wait for.

status
The status variable contains the exit status of the most recently run command (except built-in commands). It is useful for testing whether commands have succeeded or not.

shell
The shell variable is the path name of the program to be executed when a shell script file is executed that starts with a '#' character. If a command script does not begin with an '#', the program '/bin/sh' will be executed.

Changing the value of this variable will also change the value of the SHELL environment variable.

user
The user variable contains the username of the current user.

Changing the value of this variable will also change the value of the USER environment variable.

prompt
The prompt variable contains the string that the shell will print when it is ready to accept input. The default prompt string is '%' for a normal user, '#' for the superuser, and '$' for the 'bin' user. If an exclamation mark character '!' is included as part of the prompt it will be expanded into the current history list command number. The exclamation point must be escaped.

history
The history command allows you to set the number of commands to be saved in the history event list. The history events range from '1' to '75' with a default of twenty.

cdpath
The cdpath variable contains a list of directory names to use when changing directories (either 'chdir', 'cd', or 'pushd' (see below)). The cdpath is not examined if the name to change to begins with '/', './', or '../'.

If the first item in the cdpath variable is a '1', the shell will display the final directory name if the change is successful and the cdpath variable was used to get there.

cwd
The cwd variable always contains the name of the current working directory. This is the value displayed by the commands 'wd' and 'pwd'.

mail
The mail variable contains the name of the user's system mail file. The shell will check for this file when the user first logs in and will inform the user if it exists.

The default mail variable name is:
/usr/spool/mail/[username]
This check is done after sourcing the '.login' file so that the user may change the variable value if desired.

Toggle Variables

verbose
When set, the shell will echo every command before it is placed on the history list.

echo
When set, the shell will echo every command before it is executed.

ignoreeof
When set will not allow the current shell to be terminated upon typing a control-d. This is most often useful only in the login shell and is usually set in the '.login' file.

noclobber
When set, noclobber affects i/o redirection. If a redirection is creating a file, the file must not already exist. If the redirection is appending to a file or reading from a file, the file must exist. To override these checks while the noclobber variable is set, the redirections must be forced using the exclaimation point, as in '>!' or '>>&!' to redirect the standard output to a file that already exists or to append the standard error to a file that does not exist.

nonomatch
When set, it is an error for a filename pattern (see Filename Expansion below) to not match any filenames.

noglob
When set, the filename expansion characters lose their special meanings and no filename expansion is done.

set and unset

There are 2 commands for dealing with variables, 'set' and 'unset'. 'set' initializes or displays the values of variables, and 'unset' clears them. The syntax of these commands is:

set [name] [[=] value]
'set' with no arguments will display the value of all currently set variables. With 1 argument, the value of that variable (if it is defined) will be displayed. If the variable name is not currently defined, it is assumed to be a toggle variable and it is 'set'.

If there are 2 or more arguments, they are assumed to be the name and value of the variable and it is set accordingly. The equals sign (=) is optional. If the value is more than 1 word or contains other characters special to the shell, it must be enclosed in quotes.

unset name
The variable named is cleared. If it is a toggle, it's state is changed to 'unset'.

Variable substitution

Variables are introduced into the shells input stream by preceeding the variable name with the special characters '$' or '%'. These characters lose their special meaning if they are not followed by a letter or if they are escaped with a backslant '\'.

When a variable name is found in the input stream, the shell replaces the variable name (including the '$' or '%') by the word or words that are the value of the variable. Toggle variables are replaced by '1' if set, and '0' if unset.

As an example, suppose a user typed the following command:

```
set myname = "my name is John"
```

If he were then to type:

```
echo $myname
```

the program 'echo' would respond:

```
my name is John
```

(the program 'echo' simply prints it's arguments to the terminal. See 'echo (cmnd)' for more information).

If quotes (either single or double) are to appear in the value of the variable they must be escaped with backslants.

REGULUS file system and file names

The REGULUS file system is a tree-structured hierarchy of directories and files. This means that a directory may contain both plain files and other directories. Directory and file names may be from 1 to 12 characters in length. Any characters may be used in names with few restrictions, however remember that shell special characters must be escaped if used in a name. Names that begin with the character '.' are treated specially by the shell and other programs. See the next section on file name expansion for more on this.

At any given time, a user has a 'current working directory' (see 'cd (cmnd)' and 'pwd (cmnd)' for more on this). Names that do not start with the character '/' are assumed to be relative to the current directory. Names that do start with a '/' are assumed to be relative to the root of the entire file system. When giving a path name to a file, directory names are separated from each other by the '/' character also. Examples of file names are therefore:

```
/dir/user/file
file          (if current dir is '/dir/user')
user/file     (if current dir is '/dir')
```

There are 2 shorthand names in every directory, '.' and '..', that are created when the directory is made. The name '.' means the current directory and the name '..' means the parent of the current directory (that is, the directory 1 up from the current one). These names are often useful in getting files from a directory without knowing the name of that directory. For example, all of the following names refer to the same file:

```
./file          (if current dir is '/dir/user')
file            (if current dir is '/dir/user')
../user/file    (same current directory)
./../user/file  (this can get ridiculous)
```

Filename expansion

The shell provides a set of characters to be used in pattern matching of directory and file names. If a name contains any of the characters

```
* ? [
```

or begins with the character '~' it becomes a candidate for pattern matching. (The '~' is not really a pattern character, but more like an abbreviation). The function of each character is as follows:

* The asterisk '*' matches any number of any character

(including zero).

? The question mark '?' matches any single character.

[] The square brackets are used to define a character class. A list of characters inside brackets will match any one of the characters. A match is found when the first character to match is encountered. A range of characters may also be specified by giving the bounds characters separated by a dash '-'. As an example, the following patterns match all names that begin with one of the letters 'c', 'd', 'e', 'f' or 'z':

```
c* d* e* f* z*
[cdefz]*
[c-fz]*
[c-f]* z*
```

Pattern characters may appear in any position in the word. All other characters (with the exception of other shell special characters) match themselves explicitly. In addition, the character '.' at the beginning of a name and the '/' that separates path names must be matched explicitly. Note also that if the nonomatch variable is set it is an error for a pattern to not match at least one file name. In addition, in a command list with filename patterns, it is an error if all of the patterns after expansion failed to match at least one name.

~ The tilde character, when used as the first character in a name, is an abbreviation for a user's home directory. A home directory is the directory that a user is in when he first logs into REGULUS. When not followed by a letter it refers to the current users home directory as reflected by the 'home' variable. This variable is set up for the user by the shell when he logs in but may be changed by the user if he so desires.

When followed by a name, the tilde refers to the home directory of the user whose username matches the string. For example, if 2 users have usernames 'my' and 'your', and home directories '/dir/mydir' and '/dir/yourdir', and the current user is 'my', then

```
~           expands to /dir/mydir
~/file      expands to /dir/mydir/file
~your       expands to /dir/yourdir
~your/file  expands to /dir/yourdir/file
```

The shell sorts the results of each filename expansion before placing the words on the input list.

Alias substitution

The shell contains an alias facility to allow the user to customize his command environment in a convenient way. An alias is simply another way of expressing a command. Aliases are most often used to construct a shorthand command that is executed frequently, or to change the name of a command to something easier for the user to remember.

The user creates and maintains aliases with the commands 'alias' and 'unalias'.

alias [name] [[=] [value]]
With no arguments, the alias command will print the names and values for all current aliases.

If there is 1 argument it is assumed to be the name of an alias and the current value of that alias is displayed.

With 2 or more arguments, an alias is created for the given name and it will have the given value. The equals sign '=' is optional.

If the value contains shell special characters it must be enclosed within quotes to prevent their immediate interpretation.

unalias name
The unalias command removes the alias for the name from the alias list.

After a command list has been parsed, the first word (which normally is the command) is checked to see if it has an alias. If it does, the text of the alias is inserted into the command list in place of the word and the command list is again checked for an alias. If the first word of the alias value is the same as the alias name, the shell attempts to prevent looping when the alias is expanded. It is not always successful. An undetected loop will eventually run the shell out of command stack space and the command will be aborted.

A few examples might be useful for new users:

```
alias ll "ls -l"
ll mydir    (would expand into)
ls -l mydir

set lastwd "~/adir/work"
alias work "cd $lastwd ; pwd ; ls | mc"
work        (would expand into)
cd ~/adir/work ; pwd ; ls | mc
```

This would give a user a method of getting back to a

directory where he normally does his work, seeing the directory name and a list of all the files in it, by only typing 4 characters instead of the 30 that the entire command uses.

At this time, it is not possible to introduce arguments to an alias.

Built-in commands

The shell has a number of built-in commands. These are commands which normally affect the workings of the shell and need to be done in the shell's memory space. Some of them have been mentioned, and will be listed here only for completeness.

alias
alias name
alias name [=] value
See 'Alias substitution'.

cd
cd name
chdir
chdir name
Change the current working directory to the directory name specified. If no name is given, change to the directory in the 'home' variable.

If the name does not begin with '/', './' or '../' and there is no subdirectory of the current directory with the given name the shell examines the cdpath variable for a list of other directories to try. For each directory it contains, the shell constructs a pathname of the directory and the name the user typed and attempts to change to that directory.

If this fails, the shell will then check for a variable with the same name as the user typed. If there is a variable defined with this name and the value begins with the character '/', the shell attempts to change to it. Only after all of this fails will the shell report an error.

cwd
pwd
wd
Prints the current working directory.

def
A synonym for 'set' to maintain compatibility with an older shell. See 'set' for more information.

dirs
Display the current contents of the directory stack.

echo [-n] [args]
The echo command prints the arguments to the standard output, separated by a single space. If the -n flag is given, the line is not terminated with a newline.

end
See the description of 'while' below.

exec command
The given command is executed in place of the shell. There is no return if the command succeeds.

exit
exit status
logout
The current shell will exit at once with status from the 'status' variable. In the second form, the given status will be used.

glob args
The command line is expanded and the results are printed to the standard output with each word separated by a null. This is used by programs that desire the shell to perform filename expansion for them.

goto label
The shell goto is not currently implemented.

history
history #
history -r
history -r #
The history command allows the history event list to be displayed in various manners. 'history' will display the current list. 'history -r' will display it in reverse order (most current command first), and with a numeric argument it will limit the display to the last '#' items in the history list.

if expr command list
The given expression is evaluated and it's truth value established. If it is true (1) the command is executed. The following simple expressions are currently allowed:

File tests of the form '-?' where the '?' is one of the following permissions or attributes:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero length

```
f       plain file
d       directory
s       special file
```

The expression may be preceeded by the character '!' to indicate the logical 'not' of the expression.

A few examples might be useful:

```
if -r file rm file    (if file is readable, remove it)
if ! -z .mail peek .mail
  (if the file .mail is not empty, read it with peek)
if -d name dir name    (if name is a dir, list it)
```

If tests may be part or all of an alias substitution. In this way complex commands can be constructed without having to resort to 1 or 2 line command files and so is much more efficient.

login
login name
The program '/bin/login' is executed in place of the shell. If a name is given it is passed to login. See 'login (cmnd)' for more information.

newgrp
newgrp name
The program '/bin/newgrp' is executed in place of the shell. If a name is given it is passed to newgrp. See 'newgrp (cmnd)' for more information.

nice #
nice # command
The first type is used to change the priority of the currently running shell to the value given. The second form will run the given command at the specified priority. REGULUS priorities go from 0 to 255. Only the superuser may set priorities higher.

nohup
nohup command
The first form instructs the current shell and all future commands to ignore the hangup signal (see 'ssignal (sys)'). The second form will run the given command immune to hangups. Remember that commands run in the background via '&' are automatically immune to hangups.

onintr
onintr label
The onintr command is currently unimplemented.

popd
Pop the top directory name off the directory stack and change to that directory.

pushd name
pushd
In the first case, push the value of the cwd variable onto the directory stack and then change to the directory name given. The cdpath variable is examined and used if needed. In the second case, the current directory is exchanged with the next directory on the directory stack.

The directory stack is currently limited to 11 directories.

repeat cnt command
Executes the specified 'command', 'cnt' times. I/O redirection is applied only once at the beginning of the repeat.

set
set name
set name [=] value
See 'Variables' above.

setenv
setenv name
setenv name value
These commands are similar to the 'set' commands except that the variables are set in the environment (see 'environment (misc)' for more on the environment). The first form displays the current environment. The second form displays only the named variable.

The last form sets the named variable to the value given. Environment variable names are always in uppercase letters and the shell will translate lowercase variable names.

shift
shift n
The arguments of the shell are shifted left 1 position. If a number is given, the shifting will start with that argument, leaving arguments before that one untouched. ('shift' is the same as 'shift 1'). This has the effect of making the argument that was in $2 now appear in $1.

source file
The specified file is read and the commands it contains are executed inside the current shell. In this way commonly used 'set' and 'alias' commands may be created without typing them in each time.

The shell will automatically 'source' a file named '~/.nshrc' if one exists each time it is started. In addition, if the shell is a login shell (i.e., the

first shell started upon logging in) it will also source the file '~/.login' if it exists. Upon termination of the login shell a file named '~/.logout' will be sourced.

Source commands may be nested up to 4 deep. An error during a source will terminate all nested source commands.

sync
The shell will invoke the 'sync (sys)' system call to update the disk.

umask
umask #
The first form displays the current user's umask value. The umask is used when files are created (see 'umask (sys)'). The second form changes the current umask to the octal value given.

unalias name
See 'Alias substitution' above.

unset name
See 'Variables' above.

unsetenv name
This removes a variable from the environment. The name is translated into all uppercase if needed.

version
Print version information about the current shell along with the name the shell was called with.

wait
wait pid
In the first case, the shell will wait for all child tasks to complete. If a child's process id is given, the shell will wait for just that process to complete. An interrupt will terminate the waiting and force the prompt.

while
The while command is currently unimplemented.

History Commands

The shell maintains a list of the last twenty commands which the user has executed at the command level. This number may be changed up to a system dependant maximum by setting the value of the history variable.

The built-in command 'history' is used to display the commands in the list. For example:

```
% history
    1    cat test.c
    2    ls *.c
    3    cd /usr/regulus
    4    history
```

To re-execute a command from the list, type an exclamation point followed by the history list command number. For example, to re-execute the 'ls' command above you would type '!2' followed by a carriage return. The last executed command is also recognized by the shorthand of two exclamation points '!!'.

History commands can be put into the line buffer for alteration using the REGULUS line editor. There are two ways to do this. The first method is similar to that used by the editor 'ed (cmnd)'. A history command is given followed by the letter 'e' (for edit). The command is inserted into the line editor buffer, echoed to the screen, and the user is placed at the beginning of the line. For example, '!2e' will place the second command into the line editor.

An alternate method is to use the at-sign character '@' followed by the number of the command you wish to alter. For example, the same command could be placed into the buffer for editing by typing '@2'. The commands '@!' and '!!e' are equivalent and will edit the last command.

Searches through the history list can also be done on character strings. Commands of the form '!string' or '@string', where string is one or more characters in length (with a non-numeric first character), will search through the history list in reverse order for a command which matches the string (ie. '!l' would find the "ls *.c" command). Commands of the form '!?string' or '@?string' will search for the matching string any where in a history command (ie. '!?reg' would find the "cd /usr/regulus" command above).

Specific arguments from previous commands may be extracted and used. This is done by typing a history command followed by a colon ':' and an argument specifier. The following argument specifiers are allowed:

```
n    where n is an argument number.
^    the first arg (i.e. :1)
*    all arguments except 0 (the command)
n*   arguments from 'n' to the last argument.
*n   arguments from 1 to 'n'.
n-m  arguments from 'n' to 'm'.
$    the last argument
```

History commands are always echoed to the terminal before they are executed.

Command files (shell scripts)

Command files are a way to group a set of frequently used commands into a new command. It is therefore possible to construct an almost infinite number of commands using only the basic building blocks already provided by REGULUS. A command file consists simply of a file containing the commands exactly as they would be typed after the shell prompt. Any file of shell commands may then be executed by using the file name as an argument to the shell. For example, a command file to compile, load and execute a program could be:

```
cc -L prog.c
mv a.out prog
prog arg1
```

If these lines were put in a file called 'docc', then a command like:

```
sh docc
```

would perform the commands just as if they had been typed.

If the same commands are to be done many times, it would be better to set the execute permission bits on the command file. In this way, only the command name is needed since the shell will determine that the command is a command file and will invoke the shell automatically. See 'chmod (cmnd)' for more on changing a files permission bits.

The preceding example is too simple to require a command file. It would be more efficient to use an alias in this case. However, the same command file can be made more general using the argument processing of the shell. The shell variables '1', '2', etc. up to '9' are loaded by the shell with the arguments to a command file. In this way, a general purpose compile, load and execute command file can be built as follows:

```
cc -L $1.c
mv a.out $1
$1 $2 $3
```

If these commands were placed in the file named 'docc' and that file made executable, compiling and running programs would be done as:

```
docc prog argl    (this is the same as the first example)
docc newprog argl
```

At the present time, it is not possible to create aliases that use arguments, so command files are the best way to create new commands in this manner.

FILES

.nshrc	source file for initial user commands
.login	source file for login shell only
.logout	source file after exit of login shell
/etc/passwd	for finding home directories

DIAGNOSTICS

The following error messages are issued by the shell:

can't open [file]	
[name] : permission denied	The file 'name' was found but was not executable by the user
[name] : can't find	
no match	A filename pattern did not match any names
Nested parenthesized commands not allowed	
too many args : [command]	To a built-in shell command
no user [name]	In a home directory search
variable too long	Built-in variable value exceeds it allocated storage
unmatched '{'	
unmatched quote	
mismatched parens	
can't fork	To do a () command list
can't maketask [file]	Program could not be executed
can't creat [file]	
can't open pipe	No pipelines are available
invalid input to pipe	
invalid redirection	
can't use recursive aliases	
can't alias [alias,unalias]	The words 'alias' and 'unalias may not be aliased
can't source [file]	Source files nested too deeply
can't login	Program '/bin/login' could not be found
can't chdir [file]	name either does not exist, is not a directory or is protected from the user

unknown file expression specifier in an expression,
a dash '-' was not followed
by one of the valid
file test characters

SEE ALSO
maketask (sys), a.out (files)

BUGS
I/O redirection is done very early in the scan, consequently files may be opened during an 'if' even if the expression is false. Parenthesized commands can not occur on non-MMU systems.

PROGRAM
size - print size of an executable program

USAGE
size [-f] [file ...]

FUNCTION
For each filename in its argument list, 'size' will give the following information:
Name of the file, whether the file is shared text, split Instructions/Data or stand alone, size of the text of the program in decimal bytes, size of the data portion of the program in decimal bytes, and size of the uninitialized portion of the program in decimal bytes. This is followed by the sum of these three sizes in decimal, and then in hexidecimal. If the stack size or break size is not the default, zero, they are printed out. If the '-f' flag is specified, all of the above information is printed, plus whether relocation bits are present. If there are no arguments, the file a.out is used by default.

FILES
a.out - if no other argument

DIAGNOSTICS

SEE ALSO
ld (cmnd)
cc (cmnd)

BUGS

PROGRAM
sleep - delay execution

USAGE
sleep nseconds

FUNCTION
'sleep' is used to cause a delay in program execution for a specified period of time. The argument is the number of seconds for which a sleep is desired. For example:

```
sleep 10; test
```

would cause test to be run after a period of ten seconds.

The number of seconds must be greater than or equal to 0.

SEE ALSO
signal(sys)

NOTES
The maximum amount of time that can be slept is currently 32767 seconds.

PROGRAM

sort - merge and output in order

USAGE

sort [-dimr] file ... [-o outfile]

FUNCTION

'sort' merges lines from all the named files, writing the result to the standard output sorted in ascending order. If there are no files named the standard input is read and sorted. The standard input can be included in the files to be sorted by specifying the name '-'.

Order is determined by the leftmost character position in which two lines differ. The order of individual characters is determined by their ascii representation. A line which is a prefix of another line preceeds that line in sort order, due to null terminated strings.

The following flags affect the nature of the sort:

-d Dictionary sort, only blanks, letters and digits are significant. Also A<a<B<b<C... instead of A<B<C...<Z<a<b...

-i Causes 'sort' to ignore characters in the ASCII range 040 to 0176 when doing comparisons.

-m The input files are assumed to be sorted and are simply merged together in ascending order. Strange orderings result if the input files are not sorted.

-r Reverse the sense of the ordering

The -o options allows directing the output to a file rather than the standard output. The output filename may be the same as one of the input filenames.

FILES

so???????? and SO??????? - temporary files

DIAGNOSTICS

SEE ALSO

PROGRAM
split - split a file into multiple files

USAGE
split [-#] [file [name]]

FUNCTION
'split' takes the specified file, or standard input if no file is specified and splits it into smaller pieces. The file will be split into 1000 line chunks unless another number is specified. The new files will be named x?? where the ?? are the combination of two alphabetic characters starting at 'aa' and proceeding alphabetically (eg. ab, ac .. az, ba...). If another 'name' is specified then the extension will be tagged onto that.

FILES
x?? - files yielded unless another name is specified.

PROGRAM

strings - find strings of characters in files

USAGE

strings [-] [-o] [-#] [file...]

FUNCTION

'strings' looks for strings of ascii characters in the specified files or from standard output. A string is defined to be some minimum number of printable ascii characters followed by a null or a newline. The default minimum is 4 characters, but this can be changed by using the '-#' flag, where '#' is some number.

The octal offset into the file at which the string was found will be printed if the '-o' option is specified. 'strings' will only look in the data section of an a.out file unless the minus option ('-') is in effect.

'strings' is helpful for looking at unknown binary files or for looking at directory files.

FILES

a.out default input file

SEE ALSO

od (cmnd)

BUGS

PROGRAM
strip - remove symbols and relocation bits

USAGE
strip [-s] [-r] file ...

FUNCTION
'strip' removes the symbol table and/or relocation bits from executable files created by 'ld (cmnd)'. If the '-r' option is given, the relocation bits are removed and the symbol table is left. If the '-s' option is given, the relocation bits are left and the symbol table is removed.

FILES
_st?XXXXX temporary file

DIAGNOSTICS
Failures in reading the source file or writing the temporary file will be reported and the file is not changed.

SEE ALSO
cc (cmnd), as (cmnd), ld (cmnd)

BUGS
Will not strip files that do not reside in the same file system as the current directory. If the symbols have already been striped the relocation bits can not been stripped using the -r flag (ie. must not give any arguments).

PROGRAM

stty - set terminal characteristics.

USAGE

stty [-a] [-g] [options]

FUNCTION

"Stty" sets the options given in its argument list on its current output file. If no arguments exist, 'stty' reports the status of the options of its current option file. If the -a option is given, all options are reported, otherwise only selected options are given. If the -g option is given, the output is formatted so that it may be used as an argument to another stty command.

The following options are those recognized by stty, those that are preceeded by '(-)' will also be recognized as '-option' to turn that option off.

Control Modes (depending upon hardware):

0	0 baud (hang up)
50	50 baud
75	75 baud
110	110 baud
134	134.5 baud
150	150 baud
200	200 baud
300	300 baud
600	600 baud
1200	1200 baud
1800	1800 baud
2400	2400 baud
4800	4800 baud
9600	9600 baud
19200	19200 baud
38400	38400 baud
exta	External A (19200 baud)
extb	External B (38400 baud)
cs5	5 bit characters
cs6	6 bit characters
cs7	7 bit characters
cs8	8 bit characters
(-)parenb	enable parity
(-)parodd	select odd parity
(-)hupcl	hang up on last close
(-)hup	same as hupcl
(-)cstopb	use two stop bits per character
(-)cread	enable reciever
(-)clocal	enable modem control
(-)exopen	exclusive open only

Input modes:

(-)ignbrk	ignore break on input
(-)brkint	signal INTR on break
(-)ignpar	ignore parity errors
(-)parmrk	mark parity errors
(-)inpck	enable input parity checking
(-)istrip	strip input characters to seven bits
(-)inlcr	map NL to CR on input
(-)igncr	ignore CR on input
(-)icrnl	map CR to NL on input
(-)iuclc	map upper-case to lower case on input
(-)ixon	enable START/STOP output control
(-)ixany	allow any character to restart stopped output
(-)ixoff	enable START/STOP input control

Output Modes:

(-)opost	post-process output
(-)olcuc	map lower-case to upper-case on output
(-)onlcr	map NL to CR-LF on output
(-)ocrnl	map CR to NL on output
(-)onocr	do not output CR at column zero
(-)onlret	NL performs CR function
(-)ofill	use fill characters for delay
(-)ofdel	fill character is DEL (NUL)
cr0	no carriage return delay
cr1	delay of approximately .10 sec
cr2	delay of approximately .20 sec
cr3	delay of approximately .30 sec
nl0	no line feed delay
nl1	delay of approximately .10 sec.
tab0	no tab delay or translation
tab1	delay of approximately .10 sec.
tab2	delay of approximately .20 sec.
tab3	translate tabs to spaces
xtabs	translate tabs to spaces
bs0	no backspace delay
bs1	delay of approximately .10 sec.
ff0	no form feed delay
ff1	delay of approximately .10 sec.
vt0	no vertical tab delay
vt1	delay of approximately .10 sec.

Regulus line editor modes:

(-)ledit	enable line edit
(-)echo	echo every character typed
(-)echoe	echo erase as BS-SP-BS
(-)echok	erase line rather than BACKSLASH-NL
(-)isig	enable INTR and QUIT processing
(-)noflsh	do not flush after INTR or QUIT
tabset 'n'	set tab stops to every 'n' characters

Standard Unix Line Discipline (Not implemented)

(-)isig	enable INTR and QUIT processing
(-)icanon	enable canonical input processing
(-)xcase	canonical upper/lower case processing
(-)echo	echo every character typed
(-)echoe	echo erase as BS-SP-BS
(-)echok	echo kill as NL
(-)lfkc	same as echok
(-)echonl	echo NL if echo turned off
(-)noflsh	do not flush after INTR or QUIT

Control Assignments:

erase 'c'	set erase character to 'c', control characters may be specified by "^X" to get CNTRL-X. "^?" is DEL (a.k.a. Rubout) and "^-" is undefined.
kill 'c'	set kill character to 'c'
intr 'c'	set interrupt character to 'c'
quit 'c'	set quit character to 'c'
eof 'c'	set end of file character to 'c'
eol 'c'	set end of line character to 'c'
min 'n'	set min characters read to 'n', 0 <= n < 127
time 'n'	set timeout to 'n'
line 'n'	set line discipline to 'n', 0 = Unix, 1 = REGU
unix	set line discipline for Unix line discipline
regulus	set line discipline for REGULUS line disciplin

Aggregate Modes:

even(p)	parenb + -parodd + cs7
-even(p)	-parenb + cs8
odd(p)	parenb + parodd + cs7
-odd(p)	-parenb + cs8
parity	parenb + -parodd + cs7
-parity	-parenb + cs8
raw	-icanon + -isig
-raw	icanon + isig (cooked)
cooked	icanon + isig
-cooked	-icanon + -isig
cbreak	-icanon + isig (EOF character is not processed
-cbreak	icanon + isig (cooked)
lcase	-iuclc + -olcuc
LCASE	-iuclc + -olcuc
-lcase	iuclc + olcuc
-LCASE	iuclc + olcuc
nl	-icrnl + -onlcr
-nl	icrnl + onlcr + -inlcr + -igncr + -ocrnl
sane	set tty settings to something 'reasonable'

FILES

DIAGNOSTICS

SEE ALSO
ioctl (sys), tty (dev), ledit (misc)

BUGS

PROGRAM

su - become another user

USAGE

su [name]
su name [-c command [arg ...]]

FUNCTION

'su' allows one to become another user by specifying their login name and then typing the appropriate password. If the password is correct, a new shell is executed with the user ID set to that of the specified user. The default login name is root.

Additional arguments are passed to the new shell, allowing single command files or commands (-c) to be executed under the new user ID. When there are additional arguments, /bin/sh is used as the shell. Otherwise 'su' uses the shell specified in the password file.

The super-user does not need to specify a password in order to become another user.

FILES

/etc/password

SEE ALSO

login (cmnd)
newgrp (cmnd)

BUGS

PROGRAM
sum - sum the words in a file

USAGE
sum [-r] file...

FUNCTION
'sum' computes a 16-bit checksum on the specified files. It lists the files blocksize and checksum. If the '-r' option is specified a standard 16-bit crc will be computed instead. This command is generally used to test the validity of a file which has been transferred across an unreliable communication line.

DIAGNOSTICS

SEE ALSO
wc (cmnd)

PROGRAM

tail - display the last portion of a file

USAGE

```
tail [ +#[lbc] ] [file...]
tail [ -#[lbc] ] [file...]
```

FUNCTION

'tail' is used to display the end lines in a file. The default action is to print the last 10 lines in the file.

The number of units printed is modified with a flag argument beginning with a '+' or a '-'. The '+' means from the beginning of the file, and '-' from the end. The units may be specified as 'l'ines, 'b'locks, or 'c'haracters. Units are computed as lines by default.

SEE ALSO

head(cmnd)

BUGS

PROGRAM
tar - tape archiver

USAGE
tar -{cprtux}[fbndilmsvw#] [parameters] [file...]

FUNCTION
'tar' copies files to and from devices according to specified options. A key with or without the optional preceding minus '-' must be specified. The options consist of function keys and modification keys. One function key may be used along with any combination of modification keys. If no function key is specified, the contents of the tape will be printed.

Multiple volumes are used if necessary. The user can specify the number of blocks per volume. If the number of blocks per volume is not specified, 'tar' will request a new volume when it detects an end of file due to a read or write failure. If the user types an x to the prompt "mount next volume..." or "mount first volume..." 'tar' will exit.

Function keys:

- c Create a new tape. The named files are copied to the tape. If a directory is specified then all of the files in the specified directory and its subdirectories are recursively included.
- r Add files to end of the tape.
- t List the contents of the tar tape. When the 'v' modification key is also used more information about the files is displayed.
- u Update the tar tape. Add the specified files to the tape if they do not exist on the tape or if they have been modified more recently than those on the tar tape.
- x Extract the named files from the tape. If a named file matches a directory whose contents are on the tape the whole directory is recursively extracted. If a file occurs more than once on the tape, the last version overwrites any earlier ones. Files retain their modification time if the 'm' modification key is not specified. The mode is masked by the current UMASK. If the user is the super-user, nodes may be created for block and character devices.
- p Extract the named files as described for the 'x' function, ignoring the current UMASK. The original file protection modes and original file owner id's are retained if possible.

Modification keys:

- b Set block factor for tape records to the next argument. The default block size is 20 (512 byte

blocks) and the largest allowed block size is 20. This should be used consistently when updating and adding to the end of an existing tape file.

f Alter the default archive from /dev/mt? to the next specified argument. A minus sign is interpreted as standard input or standard output depending on which is appropriate.

i Interactively question the user before overwriting a file. A 'y' response will overwrite the file. Any other response will be taken as a non-confirmation and the proposed action will not take place.

l Suppress error messages generated because of unresolvable links.

m Set the modification date and time of the specified tar files to the current date and time.

s Swap bytes after reads and before writes to the tape archive. This allows for reading or writing byte swapped 'tar' files, such as those made on PDP11 or VAX11 UNIX.

v Prints the 'tar' actions to the screen in the form of a function letter followed by file name. 'tar' normally does not print out anything.

w Confirm all 'tar' actions. Before any 'tar' action takes place, the function letter followed by the file name will be printed out. If the first character given to 'tar' is a 'y', the action will take place. If the character typed is an 'x', 'tar' will terminate, and return the user to the shell.

0-7 Change the tape drive. The default tape drive is 0.

j Skip over junk headers. When a bad header is detected its name is printed and tar skips blocks until it finds a valid header. After a valid header is found tar prints the number of blocks skipped and their offsets from the beginning of the volume.

n Set the number of blocks per volume to the next argument. Note that this number is in REGULUS blocks (see the 'b' flag). REGULUS blocks are currently 512 bytes each.

d Change directory to the directory specified by the next argument prior to performing the requested function. The following pipeline can be used to move a hierarchy from the current directory to a destination directory (dest):

```
tar -cf - . | tar -xvfd - dest
```

Parameters:

Some of the modification keys ('fbnd') require an additional parameter. These parameters must appear after the keys and before the files. The order of the

modification keys specifies the order in which the parameters must be specified.

FILES
/tmp/tar????? - temporary file
/dev/mt? - tape drive
/bin/mkdir - used to create directories during extracts

DIAGNOSTICS

SEE ALSO
tar (files), ar (cmnd), umask (sys)

BUGS
Filename path lengths are restricted to 100 characters in length. The update features (-r or -u) are not implemented.

PROGRAM

tee - cause output to be directed in multi-directions

USAGE

tee [-a] [-i] [file...]

FUNCTION

'tee' causes the standard output to be written to one or more files before being passed on. One usage of this would be to have the output of a program be sent to the screen as well as to a file, so that you could watch it while it is running. For example :

```
as -p file.c | tee prfile
```

If the '-a' flag is used the output will be appended to the specified files if they already exist. The '-i' flag will cause interrupts to be ignored.

'tee' will output up to ten files.

SEE ALSO

BUGS

PROGRAM
time - time a command

USAGE
time command [arguments]

FUNCTION
Time executes the specified command and displays statistics as to the quantity of real time, user time and system time which were required.

'time' searches the path specified in the PATH environment variable for the command to execute.

The execution time is displayed in units of minutes, seconds and hundredths of seconds. The output might look like:

```
real    0:02.00
user    0:00.07
sys     0:00.15
```

SEE ALSO
times (sys)

BUGS

PROGRAM
touch - update last modified date of file

USAGE
touch [-c] file...

FUNCTION
'touch' will change the modified date of the specified files to the current time. If any of the specified files do no exist, 'touch' will try to create them, unless the '-c' flag is designated on the command line.

SEE ALSO
utime (sys)

BUGS

PROGRAM

tr - translate characters

USAGE

tr [-cds] [string1 [string2]]

FUNCTION

'tr' copies its standard input to its standard output, substituting characters found in string1 with the corresponding character in string2. The following options are permitted:

-d Delete characters that are in string1.

-c Use the complement of string1 instead of string1.

-s Replace sequences of identical characters from string2 with single characters.

Two abbreviations are helpful for specifying ranges of characters: '[A-Z]' means the string of ascii characters from A to Z inclusive and '[a*n]' means n repetitions of a. n can be octal (first character 0) or decimal, or it can be omitted. Omitting n implies enough repetitions to match the rest of the characters in string1.

The escape character \ can be used to remove special meaning from characters such as [, or it can be used to specifiy control characters. An escape character followed by one to three octal digits is interpreted as the character whose code is given by those digits.

For example, the command:

```
tr [A-Z] [a-z] < a > b
```

can be used to copy file a to file b, converting upper case letters to lower case letters.

SEE ALSO

BUGS

Nulls are always deleted.

PROGRAM
tsh - tiny REGULUS shell (command line interpreter)

USAGE
tsh [script-file]

FUNCTION
'tsh' is a command interpreter for small computer systems running REGULUS. 'tsh' is a very basic command interpreter that does not have as many capabilities as the standard REGULUS shell 'sh (cmnd)' has.

If 'tsh' is called with a filename argument, the script-file is read and the commands are executed.

When a command name is typed, 'tsh' searches for the command through the following directories:
. (the current directory)
$home/cmnd (the user's personal command directory)
/bin
/usr/bin
/etc

Pipes and I/O redirection are not implemented in 'tsh'. Background commands may be run by following the command with '&'.

The following commands are built into 'tsh':

cd [name]
chdir [name]
Change the current working directory to the directory name specified. If no name is given, 'tsh' changes to the value of the variable '$home' (see below).

login [name]
The program '/bin/login' is executed in place of 'tsh'. If a name is given, it is passed to login. See 'login (cmnd)' for more information.

exit
Causes 'tsh' to exit.

wait [pid]
If a child's process id is given, 'tsh' will wait for that process to complete. If no process id is given, 'tsh' will wait for all child processes to finish. An interrupt will terminate the waiting and force a prompt.

$prompt [new-prompt]
Change the prompt to the given string. If no string is given, change the prompt back to the default string '#' for the superuser and '%' for other users.

$error
'tsh' will display the octal value of the last error number returned by REGULUS.

$status
Causes 'tsh' to display the exit status of the most recently run command (except built-in commands).

$child
Displays the process id of the most recently created process.

$home
$home new-home
In the first case, 'tsh' will display the current home directory. In the second case, 'tsh' will change the home directory to the new directory specified. The home directory is initialy set to the value of the HOME variable from the environment.

SEE ALSO
sh(cmnd)

BUGS

PROGRAM
tty - display terminal name

USAGE
tty [-s]

FUNCTION
When called without an argument, 'tty' will return the name of the current user's terminal device. If invoked with the '-s' flag, it will not print anything, but instead will return, as its exit status, the tty number of its standard input. A -1 will be returned if the standard input is not a terminal device.

SEE ALSO
whoami (cmnd)
ttyn(subs)

PROGRAM
uname - print information about current REGULUS system

USAGE
uname [-snrvmad]

FUNCTION
'Uname' gives you information concerning the version of REGULUS you are currently running under. The following options give you particular information that is returned by uname (sys):

-s print the name of the current system (the default).

-n print the nodename (this is the name that the system is referred to by communications networks).

-r print the operating system release.

-v print the operating system version.

-m print information regarding the existence of a Memory Management Unit on this system.

-d print the root device number (in hex) and the name of the device(s) that correspond to that device number.

-a print all of the above information.

By default, 'uname' will print the system name.

SEE ALSO
uname (sys)

BUGS

PROGRAM

unrot - unrotate lines around a particular column

USAGE

unrot [column]

FUNCTION

'unrot' is a program used to help make permuted indexes for manuals, such as the one at the beginning of the REGULUS manual. 'unrot' reads from its standard input, and writes to its standard output.

'unrot' should read a file that originally was created by 'kwic' (cmnd). 'unrot' reads from its standard input and centers the lines it reads around either column 32 (the default), or the column number you give it on the command line.

The 'unrot' program will most commonly be found in a pipeline such as the following:

```
kwic omitwords < file | sort -d | unrot > INDEX
```

which will cause 'kwic' to read from the file 'file', writing to its standard output, without creating new lines for any words found in the file 'omitwords'. This command then sorts the output from 'kwic', and calls 'unrot' to center the output around the middle of the page. The file 'INDEX' will contain the permuted index.

A more complete description of how these programs interact and how to make a permuted index can be found in 'pindex (misc)'.

SEE ALSO

pindex(misc)
kwic(cmnd)
sort(cmnd)

BUGS

PROGRAM

vdir - print directory of versados volume or catalog

USAGE

vdir versados_volume [catalog]

FUNCTION

'vdir' prints directories of versados volumes and catalogs. If no catalog is specified the names of all of the catalogs on the volume are listed, otherwise all of the files in the named catalog are listed. The versados_volume can be any regulus file, normally it would be a floppy disk. The format for catalog name is [USERNO.]NAME, where USERNO is the user number and NAME is the catalog name. If no user number is specified the first catalog which matches the name is selected.

The following information is printed for each file in a catalog:

Name and extension
Write protection code (octal)
Read protection code (octal)
File attributes (octal)
Physical sector number (PSN)
of first file access block (FAB) (hex)
Record size (octal)

In some files the first character has been replaced with a null. These files have probably been deleted. In the directory listing the null is replaced with a '>'.

SEE ALSO

vget(cmnd)

BUGS

PROGRAM

vget - extract files from a versados catalog

USAGE

vget versados_volume catalog [files]

FUNCTION

'vget' extracts files from versados catalogs. If no files are specified all of the files in the named catalog are extracted. The versados_volume can be any regulus file, normally it would be a floppy disk. The format for catalog name is [USERNO.]NAME, where USERNO is the user number and NAME is the catalog name. If no user number is specified the first catalog which matches the name is selected.

The file names must match exactly, typically versados file names are uppercase only. 'vget' does not map lower case to upper case prior to comparing file names. Upper case is mapped to lower case when the file is created on the Regulus file system. The versados wildcard operator '*' is implemented, it does not operate the way the '*' wildcard operates under Regulus. It must be quoted or it will be expanded by the Regulus shell (e.g. "*.CC" instead of *.CC).

Only files with attributes (att) of 1 (sequential) or 3 (indexed sequential) and record size (recsize) of 0 (variable length) can be extracted.

SEE ALSO

vdir(cmnd)

BUGS

PROGRAM

wc - word count

USAGE

wc [[-ptv] [-b#] [-s#]] [file...]

FUNCTION

'wc' lists character, word and line counts for one or more files. It will optionally print a page count and the approximate time it takes to display the file at a specific baud rate.

p	print number of pages in file (66 lines per page).
t	print time to display (default is 300 baud).
v	print verbose, headings will be displayed.
b#	alternate baud rate.
s#	alternate page size.

If a file is not specified, 'wc' will read from standard input. If more than one file is specified 'wc' will list the total sizes for all the files as well as their individual totals.

BUGS

PROGRAM
wd - print current working directory

USAGE
wd

FUNCTION
'wd' will print the full pathname of the current working directory.

FILES

DIAGNOSTICS

SEE ALSO
cd (cmnd)
pwd (cmnd)

BUGS
This command is a built-in function of the shell.

PROGRAM

what - print version information from a file

USAGE

what [-] file ...

FUNCTION

'what' looks for a version specification in the designated files.

Version strings begin with the SCCS indentification string '@(#)'. The characters following the identifier are printed up to a null, double quote, carriage return or backslash.

All occurances of such strings are printed. 'what' searches only the data portion of an object file unless the '-' option is given.

Non-object files are always searched from the beginning.

SEE ALSO

BUGS

PROGRAM
who - list currently logged on users

USAGE
who [am I]

FUNCTION
'who' prints a list of currently logged on users, including the terminal name they are on and the time they logged in.

If there are two arguments, as in "who am I", 'who' lists information for the person who is logged in at your terminal.

FILES
/etc/utmp file of currently logged in users

SEE ALSO
init (maint)
login (cmnd)

BUGS

PROGRAM
whoami - print current user and group

USAGE
whoami [-e]

FUNCTION
'whoami' prints out the current user's name and group. The current user's actual user id is used to determine the user name. If any argument is given, for example "whoami -e", the effective user and group ids are used.

FILES
/etc/passwd
/etc/group

SEE ALSO
who (cmnd)
logname (cmnd)

BUGS

PROGRAM

write - write to another user

USAGE

write user [tty]

FUNCTION

'write' allows users to send interactive messages between two terminals. Strings of characters typed on the initiator's terminal are copied to the receiver's terminal. 'write' is terminated by entering the end of file character. 'EOF' is printed on the receiver's screen at this point.

Upon initiation of this command a message will appear on the screen of the terminal associated with 'user'. If this person is not logged in or has disallowed writing to his terminal the associated message will be printed on the executor's screen. Otherwise a message "Message from <yourname>, on tty<*>" is printed on the receiver's screen, along with the current date and time.

Shell commands can be executed by typing the command on a line by itself with an exclamation point '!' in the first column.

FILES

/etc/utmp - list of currently logged in users

SEE ALSO

who (cmnd)
mail (cmnd)
mesg (cmnd)

BUGS

NAME

access - check file permission for user

DESCRIPTION

The 'access' system call allows you to check the accessibility of a file according to the REAL user id or group id. 'access' takes two parameters. The first is the name of the file to check for access permissions. The second argument is a bit pattern that corresponds to the access permissions that you are requesting. If the second argument is zero, then you are requesting whether or not the directories to the file are searchable by the REAL user or group id, and that the file exists. If the second argument is non-zero, you are testing the logical 'or' of the following permissions that the real user or group has with the file.

```
1    execute permission
2    write permission
4    read permission
```

This system call is useful to 'setuid (sys)' or 'setgid (sys)' programs to test the real permissions of the person running the task. The superuser has all privileges as usual.

CALLING SEQUENCE

```
int access(fname, flags)
char *fname;                    // Name of file to check
int flags;                      // Bit pattern of permissions to test
```

ASSEMBLY INTERFACE

```
move    flags,-(sp)             ; Push the bit pattern
move.l  fname,-(sp)             ; Push the file name
move.l  sp,r8                   ; Pass the starting address of the params
move    #35,r0                  ; Pass the system call number
trap    #0                      ; Call the system
```

SEE ALSO

stat (sys),fstat (sys)

RETURN VALUES

A -1 is returned if the path to the file was not searchable by the user, or if the file does not exist, or the permissions requested were not granted by the file to the user. 0 is returned to note that the requested conditions do exist. Note that even though one has write permission to a directory, a 'write (sys)' to the directory will fail.

BUGS

NAME

alarm - set up a process alarm signal

DESCRIPTION

The 'alarm' system call arranges for the system to send an alarm signal to the process after the requested number of seconds has elapsed. Return from the call is immediate.

CALLING SEQUENCE

```
int alarm(seconds)
int seconds;            // Number of seconds 'till alarm
```

ASSEMBLY INTERFACE

```
move    seconds,-(sp)
move    sp,r8           ; Pass address of arguments
move    #48,d0          ; Pass system call number
trap    #0              ; Call the system
```

SEE ALSO

signal(sys), pause(sys), sleep(sys)

RETURN VALUES

'alarm' returns the remaining time until the next alarm. It will return zero if no alarm has been set.

BUGS

NAME

attach - attach user process to hardware interrupt

DESCRIPTION

'attach' takes the address of a user interrupt servicing routine and places it in the Kernel at the specified interrupt vector location. The interrupt vector location must be between 0 and 1024 and be on a word boundary. The user interrupt servicing routine gets control of the CPU without any intervention of the operating system, and the CPU is in supervisor state. Therefore, the interrupt routine is responsible for saving and restoring any temporary registers, etc. Only the super-user may make this system call. There is a system imposed maximum active attaches, typically four. When the process ends, the previous interrupt vectors are restored.

The usage of this system call is VERY strongly discouraged. Do not expect assistance in debugging programs using this system call.

CALLING SEQUENCE

```
char *attach(intvadd,uintadd)   //returns old interrupt vector
char **intvadd;                 //kernel address of interrupt vector
int (*uintadd)();               //address of user interrupt routine
```

ASSEMBLY INTERFACE

```
move.l    #uintadd,-(sp)        ;push address of uintadd onto stack
move.l    #intvadd,-(sp)        ;push address of intvadd onto stack
move.l    sp,r8                 ;current sp points to tadd address
move.l    #61,r0                ;system call number
trap      #0                    ;system call
```

RETURN VALUES

-1 if not super-user, invalid address or too many attach calls else the old interrupt vector address.

NAME

boot - stop or reboot REGULUS kernel

DESCRIPTION

The boot system call is system dependent and may or may not be available on a particular hardware configuration. Depending upon the hardware configuration, 'boot' may be able to:

1. stop REGULUS, exiting to hardware debugger
2. reboot REGULUS, using in-memory system
3. reboot REGULUS, using specified argument to boot

'boot' may only be called by the super-user.

CALLING SEQUENCE

```
int boot(cmd,arg)       returns -1 if error, no return otherwise
int cmd;                command:
                           1:  stop REGULUS, exit to debugger
                           2:  reboot REGULUS, using in-memory
                           3:  reboot REGULUS, using argument
char *arg;              optional argument for cmd=3.
```

ASSEMBLY INTERFACE

```
move.l  #arg,(sp)-      ;address of argument
move    cmd,(sp)-       ;cmd
move.l  #55,r0          ;system call number
trap    #0              ;system cal
```

SEE ALSO

boot (misc)

RETURN VALUES

-1 if not super user, cmd is invalid or arg is invalid.

NAME

brk - set program break to specific address

DESCRIPTION

The 'brk' system call allows a user task to allocate or deallocate bss memory. The old 'break' address (the highest data space address) is returned and the 'break' address is set to the value specified. If the address is greater than the old 'break', memory may be allocated to the task. This may result in the task being swapped out.

CALLING SEQUENCE

```
char *brk(addr)          returns old break address or -1 for error
char *addr;              address of new break
```

ASSEMBLY INTERFACE

```
move.l  #addr,-(sp)     ;push address of addr onto stack
move.l  sp,r8           ;current sp points to addr address
move.l  #16,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

sbrk (sys)

RETURN VALUES

The old 'break' address is returned. -1 is returned if the break address cannot be changed.

NOTES

Since the task may need to be swapped to allocate memory a real-time task cannot do 'brk' calls.

NAME

chdir - change working directory of task

DESCRIPTION

The 'chdir' system call is used to change the working directory of a task. The new working directory is passed as a null terminated string. The task must have access permission for the directory (as shown in the execute permission bits) and all of it's parent directories.

CALLING SEQUENCE

```
int chdir(dirn)          returns -1 if error, 0 otherwise
char *dirn;              new working directory name
```

ASSEMBLY INTERFACE

```
move.l  #dirn,-(sp)     ;push address of dirn onto stack
move.l  sp,r8           ;current sp points to address of dirn
move.l  #10,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

RETURN VALUES

If any of the directories are protected, a -1 is returned and the current directory is not changed. Zero is returned if there was no error.

NAME

chgrp - change group id of file

DESCRIPTION

The 'chgrp' system call changes the group id of the named file to the numerical group id given. The group id can be changed only by the owner of the file or the super-user.

CALLING SEQUENCE

```
int chgrp(fname,groupid)    returns -1 if error, 0 otherwise
char *fname;                name of file to change group
int groupid;                new group id of file
```

ASSEMBLY INTERFACE

```
move    groupid,-(sp)   ;push groupid
move.l  #fname,-(sp)    ;push address of fname onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #15,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

chown (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the file does not belong to the current user or the user is not the super-user.

NAME

chmod - change access mode bits of a file

DESCRIPTION

The 'chmod' system call is used to change the protection bits of a file. Only the owner or superuser may change the protection of a file. The protection of the named file is changed to the absolute number given. For a description of the protection bits, see 'chmod (cmnd)' or 'ls (cmnd)'.

CALLING SEQUENCE

```
int chmod(fname,mode)     returns -1 if error, 0 otherwise
char *fname;              name of file to change mode
int mode;                 new mode bits of file
```

ASSEMBLY INTERFACE

```
move    mode,-(sp)        ;push mode
move.l  #fname,-(sp)      ;push address of fname
move.l  sp,r8             ;sp points to arg block
move.l  #11,r0            ;system call number
trap    #0                ;system call
```

SEE ALSO

chmod (cmnd)
ls (cmnd)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the file is not owned by the current user or you are not the superuser.

NAME

chown - change owner of a file

DESCRIPTION

The 'chown' system call is used to change the owner of a file. Only the super-user may execute this call. The numerical ownerid of the named file is changed to the id given. Care should be taken that the new id has been assigned to a user.

CALLING SEQUENCE

```
int chown(fname,userid)     returns -1 for error, 0 otherwise
char *fname;                name of file to change owner
int userid;                 new owner id of file
```

ASSEMBLY INTERFACE

```
move    userid,-(sp)    ;push userid
move.l  #fname,-(sp)    ;push address of fname onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #12,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

chgrp (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if you are not the super-user.

NAME

chroot - change concept of filesystem's root

DESCRIPTION

'chroot' causes the filesystem to appear to be a subset of itself. The directory 'addr' will subsequently appear to be '/'. Only the super-user can execute this command.

CALLING SEQUENCE

```
int chroot(addr)
char *addr;
```

ASSEMBLY INTERFACE

```
move.l    #addr,-(sp)
move.l    sp,r8
move      #56,r0
trap      #0
```

SEE ALSO

chroot(cmnd)

RETURN VALUES

-1 is returned if the 'chroot' is unsuccessful.

NAME

close - close an open file from reading/writing

DESCRIPTION

The 'close' system call closes the file associated with the given file descriptor that was returned by a previous call to 'open (sys)' or 'creat (sys)'. The file descriptor is made available for subsequent calls to 'open' or 'creat'. There is a system defined limit to the number of files that may be open at the same time. This allows a task to have access to more than this limit of files.

CALLING SEQUENCE

```
int close(fd)          returns -1 for error, 0 otherwise
int fd;                file descriptor
```

ASSEMBLY INTERFACE

```
move.l  #fd,r8         ;move address of fd to r8
move.l  #6,r0          ;system call number
trap    #0             ;system call
```

SEE ALSO

open (sys)
creat (sys)
pipe (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the given file descriptor does not refer to an open file.

NAME

creat - create a new file

DESCRIPTION

The 'creat' system call allows a task to create new files or to rewrite old files. If a file of the specified name does not exist, a new file with the specified protections will be created. If a file of the given name already exists, it is truncated to zero size and then opened. In either case, the file is opened for both reading and writing.

The file is created with protections specified by the protection bits 'anded' with the complement of the current value of the user's umask. The protection is arbitrary and need not allow either reading or writing by anyone.

Note that fifo files can be created only by using 'mknod (sys)' system call.

CALLING SEQUENCE

```
int creat(fname,mode)    returns -1 or file descriptor
char *fname;             pointer to new file name
int mode;                initial file access mode bits
```

ASSEMBLY INTERFACE

```
move    mode,-(sp)       ;push mode
move.l  #fname,-(sp)     ;push address of fname onto stack
move.l  sp,r8            ;current sp points to arg block
move.l  #8,r0            ;system call number
trap    #0               ;system call
```

SEE ALSO

open (sys)
close (sys)
umask (sys)

RETURN VALUES

The file descriptor of the opened file is returned if there was no error. -1 is returned if the file could not be created. This can be because of too many open files, a directory was not searchable, or the user does not have permission in the directory.

NAME

dup - get a duplicate file descriptor

DESCRIPTION

The 'dup' system call allocates a file descriptor which will be a duplicate of the file descriptor given. The file descriptor must have been the result of a previous call to 'open (sys)', 'creat (sys)' or 'pipe (sys)'. Since the file descriptor allocation algorithm is to allocate the lowest possible descriptor, this call is commonly used to change the standard input, output or error file descriptors for a child task.

There is a system defined limit to the number of assigned file descriptors for a given task. Exceeding this limit will cause 'dup' to fail.

CALLING SEQUENCE

```
int dup(fd)          returns -1 if error, file desc. otherwise
int fd               file descriptor
```

ASSEMBLY INTERFACE

```
move.l  #fd,r8       ;move address of fd to r8
move.l  #18,r0       ;system call number
trap    #0           ;system cal
```

SEE ALSO

open (sys)
close (sys)
pipe (sys)
creat (sys)

RETURN VALUES

The duplicate file descriptor is returned if there were no errors. -1 is returned if the 'dup' was unsuccessful. Possible causes are the file descriptor was not associated with an open file or too many files open.

NAME

execv, execl, execve, execle, execvp, execlp - execute program

DESCRIPTION

The family of exec system calls all "overlay" the current task with the specified executable object file. All of the resources allocated to the current task are released, then the specified object file is read into memory and executed in place of the old task. All open files will be passed to the new task, except ones that have been marked by fcntl (sys) to be closed over execs. The arguments and environments specified in the exec calls are made available to the new task via the argc, argv and envp arguments in the new tasks main routine. All these system calls are calls to maketask with the OVERLAY option set.

There are two types of exec system calls: one where the arguments are passed by giving the address of an argument block (execv, execve, execvp), and the other where they are passed as a sequence of string arguments terminated by a NULL pointer (execl, execle, execlp). Note that it is VERY important that the arguments be terminated by a NULL pointer, rather than merely a constant zero, due to the difference in size between a constant zero and a pointer NULL.

For those exec calls where no environment pointer is specified, the global variable 'environ' is used by default.

The execvp and execlp calls use the environment variable 'PATH' to search for an executable file with the given path suffix. The current working directory is always searched first.

CALLING SEQUENCE

```
int execv(fname,args)
char *fname;
char *args[];

int execl(fname,arg0,arg1,...,argn,NULL)
char *fname;
char *arg0, *arg1, ..., *argn;
```

The next two system calls differ from the above two only in that the next two are also passed pointers to the environment
which may be specified by the user:

```
int execve(fname,args,env)
char *fname;
char *args[];
char *env[];
```

```
int execle(fname,arg0,argl,...,argn,NULL,env)
char *fname;
char *arg0, *argl, ..., *argn;
char *env[];
```

The next two system calls, execvp and execlp automatically use the environment variable "PATH" to search for an executable file with the given path suffix. The current working directory is always searched first:

```
int execvp(fname,args)
char *fname;
char *args[];

int execlp(fname,arg0,argl,...,argn,NULL)
char *fname;
char *arg0, *argl, ..., *argn;
```

SEE ALSO

maketask (sys)
fork (sys)

RETURN VALUES

-1 is returned if the new task could not be created. These calls only return if an error has occured.

NAME

execv - execute an executable file

DESCRIPTION

The 'execv' system call frees up the resources allocated to a task and then copies the specified object file into memory where it is executed in place of the old task. The arguments are copied to the new tasks stack so that the normal C argument passing syntax will work. The new task inherits the old tasks standard input, output and error files. In addition to files, the new task inherits the ignored signals of the old task.

This is implemented as a call to the 'maketask' system call.

CALLING SEQUENCE

```
int execv(fname,argptrs)    returns -1 if error, otherwise no ret
char *fname;                pointer to file name of executable file
char *argptrs[];            pointer to argument block
```

ASSEMBLY INTERFACE

```
move.l  #argptrs,-(sp)  ;push arg block address
move    priority,-(sp)  ;push priority
move    flags,-(sp)     ;push flags
move.l  #fname,-(sp)    ;push fname address onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #2,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

maketask (sys)

RETURN VALUES

-1 is returned if the new task could not be created.

NAME

exit - end task with specified status

DESCRIPTION

The 'exit' system call terminates the task. The status is obtainable by the parent task with 'wait (sys)'. All resources allocated to the task are freed and all open files are closed. Any output remaining in the terminal buffer is flushed. An unspecified status value causes a garbage value to be returned to the parent.

The C function 'exit' may cause cleanup actions before actually exitting. The function '_exit' circumvents all cleanup activity.

CALLING SEQUENCE

```
int exit(status)          returns - none
int status;               exit status returned to parent task

int _exit(status)         returns - none
int status;               exit status returned to parent task
```

ASSEMBLY INTERFACE

```
move.l  #status,r8      ;move address of status to r8
move.l  #1,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

wait (sys)
close (sys)
flush (sub)

RETURN VALUES

There is no possibility of this system call returning. The value passed to 'exit' is made available to the parent process through 'wait (sys)'.

NAME

fcntl - file control settings

DESCRIPTION

The 'fcntl' system call allows for the control of open files. The file is specified by a file descriptor, which is the result of a previous call to 'open (sys)', 'creat (sys)' or 'pipe (sys)'. The functions fcntl can perform are:

F_DUPFD duplicate a file descriptor no lower than the specifed argument.

F_GETFD get the open-across-exec flag for the file: If fcntl returns F_OPEN, it means REGULUS will leave the file open across execs and maketasks. If fcntl returns F_CLOSE, it means REGULUS will close the file across execs and maketasks. (F_OPEN and F_CLOSE are defined in <fcntl.h>.)

F_SETFD set the open-across-exec flag for the file: The third argument to fcntl should be either F_OPEN or F_CLOSE.

F_GETFL get the read/write, no-delay and append mode flags for the file: fcntl returns the bits as defined by O_RDONLY, O_WRONLY, O_RDWR, O_NDELAY and O_APPEND.

F_SETFL set the no-delay and append mode flags for the file: The third argument to fcntl should be either O_NDELAY or O_APPEND.

CALLING SEQUENCE

```
#include <fcntl.h>
int fcntl(fd,cmd,arg)   -1 if error, otherwise return value
int fd;                 file descriptor
int cmd;                one of above commands
int arg;                OPTIONAL argument depending upon cmd.
```

ASSEMBLY INTERFACE

```
move    arg,(sp)-       ;move arg to stack
move    cmd,(sp)-       ;move cmd to stack
move    fd,(sp)-        ;move fd to stack
lea     (sp),r8         ;move address of arg block to A0
move.l  #54,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

open (sys)
close (sys)
pipe (sys)
creat (sys)

RETURN VALUES

Depending upon the function performed:

F_DUPFD -1 for failure, otherwise valid file descriptor
F_GETFD -1 for failure, otherwise F_CLOSE or F_OPEN
F_SETFD -1 for failure, otherwise 0.
F_GETFL -1 for failure, otherwise file control flags.
F_SETFL -1 for failure, otherwise 0.

NAME

fork - create a copy of a running task

DESCRIPTION

The 'fork' system call creates a task that is an exact copy of the current task. The new task inherits copies of the old task's open files and signal environment, including caught and ignored signals.

This is implemented as a call to the 'maketask' system call.

CALLING SEQUENCE

```
int fork()                  returns 0 for child, child pid or
                            -1 for parent
```

ASSEMBLY INTERFACE

```
move.l  #argptrs,-(sp)  ;push arg block address
move    priority,-(sp)  ;push priority
move    flags,-(sp)     ;push flags
move.l  #fname,-(sp)    ;push fname address onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #2,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

maketask (sys)

RETURN VALUES

The process id of the new task is returned to the parent task. Zero is returned to the the new task so that it will know that it is the new task. -1 is returned to the parent if the task could not be created.

NAME

dup2 - Version 7 - dup to specified file descriptor

DESCRIPTION

The 'dup2' system call performs the same function as 'dup (sys)', except it copies the file descriptor to a specified file descriptor. The specified file descriptor is closed first. This function is performed by 'close (sys)' and'fcntl (sys)' in REGULUS.

CALLING SEQUENCE

```
int dup2(fd,fd2)        returns -1 if error, file desc. otherwise
int fd;                 file descriptor to dup
int fd2;                file descriptor to dup to
```

SEE ALSO

open (sys)
close (sys)
pipe (sys)
creat (sys)
dup (sys)
fcntl(sys)

RETURN VALUES

The duplicate file descriptor is returned if there were no errors. -1 is returned if the 'dup' was unsuccessful. Possible causes are the file descriptor was not associated with an open file or too many files open.

NAME

fstat - Version 7 get file status by file descriptor

DESCRIPTION

The 'stat' system call is used to get file information about the file whose open descriptor is specified. This is implemented as a function that does a REGULUS 'fstat (sys)' system call, then rearranges the returned values.

Note that the inode number is the REGULUS index record number truncated to 16 bits and the inode changed time is the REGULUS modified time.

CALLING SEQUENCE

```
#include <stat.h>
int fstat(fd,buf)       returns -1 for error, 0 otherwise
int fd;                 file descriptor of file to stat
struct stat *buf;       pointer to stat buffer

/* stat structure for Unix Version 7 */

struct stat {
    int st_dev;         /*device inode is on*/
    int st_ino;         /*inode number*/
    int st_mode;        /*file mode bits*/
    int st_nlink;       /*number of links*/
    int st_uid;         /*owner id*/
    int st_gid;         /*group id*/
    int st_rdev;        /*real device if mount, char or block*/
    long st_size;       /*file size in bytes*/
    long st_atime;      /*access time*/
    long st_mtime;      /*modified time*/
    long st_ctime;      /*inode changed time*/
};

#define     S_IFMT      0170000     /*file type*/
#define     S_IFDIR     0040000     /*directory*/
#define     S_IFCHR     0020000     /*character*/
#define     S_IFBLK     0060000     /*block*/
#define     S_IFREG     0100000     /*regular*/
#define     S_IFMPC     0030000     /*multiplexed char*/
#define     S_IFMPB     0070000     /*multiplexed block*/
#define     S_ISUID     0004000     /*set user id*/
#define     S_IGUID     0002000     /*set group id*/
#define     S_ISVTX     0001000     /*save text*/
#define     S_IREAD     0000400     /*owner read*/
#define     S_IWRITE    0000200     /*owner write*/
#define     S_IEXEC     0000100     /*owner exec*/
```

SEE ALSO

stat (v7/sys)
fstat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if

the named file does not exist or if any of the directories leading to it are unreadable due to protections.

NAME

gtty - Version 7 get terminal control information

DESCRIPTION

This returns the terminals current input/output configuration in the buffer provided. See 'stty (v7/sys)' for a detailed explanation of the tty buffer. This is implemented as an interface routine to 'ioctl (sys)'.

CALLING SEQUENCE

```
#include <sgtty.h>
int gtty(fd,ttybuf)          returns -1 for error, 0 otherwise
int fd;                      file descriptor of char special file
struct sgtty *ttybuf;        pointer to tty buffer
```

SEE ALSO

stty (v7/sys)
ioctl (sys)

RETURN VALUES

A negative number is returned if the file descriptor does not refer to an open terminal device. Zero is returned if there was no error.

NAME

ioctl - Version 7 character device control

DESCRIPTION

The 'ioctl' system call allows the user to change a terminal's current input/output configuration. This is for Version 7 ioctl system calls, which are different from REGULUS and System III ioctl system calls. It is highly recommended that programs use the new ioctl calls rather than the interface routines.

The following ioctl commands are supported:

```
TIOCGETP   get tty buffer settings
TIOCSETP   set tty buffer settings, wait for drain
TIOCSETN   set tty buffer settings immediately
TIOCEXCL   set 'exclusive use' mode
TIOCNXCL   clear 'exclusive use' mode
TIOCHPCL   hang up on last close
TIOCFLUSH  flush input/output queues
TIOCGETC   get special characters
TIOCSETC   set special characters
TIOCGETD   get line discipline setting
TIOCSETD   set line discipline setting
```

The tty interface buffer has the following structure:

```
struct tty {
    char sg_ispeed;      input baud rate (see below)
    char sg_ospeed;      output baud rate
    char sg_erase;       erase character
    char sg_kill;        kill character
    int sg_flags;        bit masks for modes
};
```

The speed bytes change the speed of the terminal if the device is capable of changes. Changing the speed on a device not so equipped or to a speed that is not implemented for that device is useless. The following table shows the correspondence between the speed settings and the actual baud rates implemented:

```
B0      0   0 baud (hang up)
B50     1   50 baud
B75     2   75 baud
B100    3   110 baud
B134    4   134.5 baud
B150    5   150 baud
B200    6   200 baud
B300    7   300 baud
B600    8   600 baud
B1200   9   1200 baud
B1800  10   1800 baud
B2400  11   2400 baud
B4800  12  4800 baud
```

B9600	13	9600 baud
EXTA	14	19200 baud
EXTB	15	38400 baud

The terminal mode word sets the way REGULUS handles certain terminal dependent operations, including character mapping, echo, tabs, etc. The bit masks and their meanings are as follows:

ALLDELAY	0177400	All delay settings bits
BSDELAY	0100000	Backspace Delay
BS0	0000000	No backspace delay
BS1	0100000	Delay approximately .10 sec.
VTDELAY	0040000	Form feed delay
FF0	0000000	No form feed delay
FF1	0040000	Delay approximately .10 sec.
CRDELAY	0030000	Carriage return delay
CR0	0000000	No carriage return delay
CR1	0010000	Delay approximately .10 sec.
CR2	0020000	Delay approximately .20 sec.
CR3	0030000	Delay approximately .30 sec.
TBDELAY	0006000	Tab delay
TAB0	0000000	No tab delay
TAB1	0002000	Delay approximately .10 sec.
TAB2	0004000	Delay approximately .20 sec.
XTABS	0006000	Translate tabs to spaces
NLDELAY	0001400	New line (line feed) delay
NL1	0000400	Delay approximately .10 sec.
NL2	0001000	Delay approximately .20 sec.
NL3	0001400	Delay approximately .30 sec.
EVENP	0000200	Even parity
ODDP	0000100	Odd parity
RAW	0000040	Raw mode, wake up on all characters
CRMOD	0000020	map LF to CR-LF, echo CR or LF as CR-LF
ECHO	0000010	Echo input
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Same as raw, but handle INTR and QUIT
TANDEM	0000001	Enable XON-XOFF control

Delay bits specify one of a number of delays. A value of zero is used for no delay. All other values are system dependent.

Raw mode indicates that each character is to be passed to the reading process as soon as it is available. No special handling of the end-of-file (EOT), interrupt (DEL), or backspace is done. The line editor must be disabled with raw mode.

Cbreak mode is like raw mode on input in that the process is awakened on each character, but it also does kill and interrupt processing. No erase or input mapping is done. All output processing is still done. The cbreak mode is incompatible with raw mode and the line editor, so only one of the three should be used

at any one time.

REGULUS will map carriage returns and line feeds into a carriage return /linefeed pair when enabled.

REGULUS will map uppercase characters into their lowercase equivalents if needed for older terminals.

The tab to blanks mapping bit is required to use other than hardware specified tab sizes.

The tty special character buffer is defined as:

```
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 (new-line)*/
};
```

CALLING SEQUENCE

```
#include <sgtty.h>
int ioctl(fd,cmd,arg)   returns -1 for error, 0 otherwise
int fd;                 file descriptor of char special file
char *arg;              optional command argument
```

SEE ALSO

stty (cmnd)
ioctl (sys)
gtty (v7/sys)
stty (v7/sys)

RETURN VALUES

Zero is returned if there were no errors. -1 is returned if the file descriptor does not refer to an open terminal device or the user does not have write permission on that device.

NAME

mpx - Version 7 multiplexed file manipulation

DESCRIPTION

The 'mpx' system call, which creates and manipulates 'multiplexed' files, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

freecomm - release named common memory

DESCRIPTION

The 'freecomm' system call releases access to a "named common" area of memory. This allows a task to execute a subsequent getcomm call to access a different named common. The named common area is not actually deallocated until the last task which has allocated it does a release. Named common areas are released when a task terminates.

CALLING SEQUENCE

```
int freecomm()              returns -1 for error
```

ASSEMBLY INTERFACE

```
move.l  #53,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

getcomm (sys)
signal (sys)
pipe (sys)

RETURN VALUES

-1 is returned if there is an error.

NAME

fstat - get file status by file descriptor

DESCRIPTION

The 'fstat' system call is used to get file information about an open file. If there are no errors, the index record number of the file is returned. The information in the given structure is available.

CALLING SEQUENCE

```
long fstat(fd,buf)          returns -1 for error, index block numbe
int fd;                     file descriptor of file to stat
struct indblk {             address of buffer to place status infor
    int     i_magic;          //magic number to id indirect blocks
    long    i_size;           //file size
    int     i_ownerid;        //file owner id
    char    i_groupid;        //group id of file
    char    i_nlinks;         //file reference count
    long    i_moddate;        //file modification date and time
    long    i_accdate;        //file access date and time
    int     i_flags;          //file type and access mode
    long    i_blkptr[2];      //first data block numbers or major/m
                              //device # for special files
    int       i_unused;
    int       i_dev;            //major/minor device # where file
} *buf;
```

The above structure exists in the '#include' file <stat.h>.

ASSEMBLY INTERFACE

```
move.l  #uadd,-(sp)     ;push address of uadd onto stack
move    fd,-(sp)        ;push fd
mov.l   sp,r8           ;current sp points to arg block
move.l  #21,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

stat (sys)

RETURN VALUES

The file index record number is returned if there was no error. -1 is returned if the file descriptor does not correspond to an open file.

NAME

ftime - get time of day (BSD compatible)

DESCRIPTION

The 'ftime' system call returns a structure with the following information: the system's idea of the time of day in seconds (see 'time (sys)') the number of milliseconds of the time of day clock, the number of minutes from Greenwich, the daylight savings time flag. This system call is compatible with BSD ftime system call.

CALLING SEQUENCE

```
#include <sys/timeb.h>
int ftime(tp)              returns -1 for error, 0 otherwise
struct timeb *tp;          pointer to timeb structure returned
```

ASSEMBLY INTERFACE

```
move.l    #tp,-(sp)        ;push address of tadd onto stack
move.l    sp,r8            ;current sp points to tadd address
move.l    #60,r0           ;system call number
trap      #0               ;system call
```

SEE ALSO

stime (sys)
time (sys)

RETURN VALUES

-1 if the specified address is invalid, 0 otherwise

NAME

getcomm - acquire access to named common memory

DESCRIPTION

The 'getcomm' system call acquires access to a "named common" area of memory. If a common area with the given name does not exist, it is allocated and its address is returned. If a common area with the given name already exists, the specified length must match the length of the allocated area and the effective owner id of the requesting task must match the owner id of the common area. If these two conditions are met, the address of the named common area is returned. Each task may have at most one active named common area at any given time. All synchronization between multiple tasks using a named common memory area must be done by these tasks using either signals or semaphores in the named common memory.

CALLING SEQUENCE

```
char *getcomm(name,length)   returns address or -1 for error
char *name;                  name of common area (first 12 bytes u
long length;                 length of common area in bytes
```

ASSEMBLY INTERFACE

```
move.l  length,-(sp)     push length of common area
move.l  #name,-(sp)      push address of name
move.l  sp,r8            ;current sp points to arg block
move.l  #52,r0           ;system call number
trap    #0               ;system call
```

SEE ALSO

freecomm (sys)
signal (sys)
pipe (sys)

RETURN VALUES

-1 is returned if there is an error. The address of the named common memory area is returned if no error.

NAME
getegid - get effective group id of task

DESCRIPTION
The 'getegid' system call returns the 16 bit effective group identification number of the current task. The effective group id may be different from the real group id if the task has the set-group-id permission bit set.

CALLING SEQUENCE
int getegid() returns effective group id of task

SEE ALSO
getgid (sys)

RETURN VALUES
The effective group id is returned.

NOTES
This is actually implemented as a call to the 'getgid (sys)' system call.

NAME

geteuid - get effective user id of task

DESCRIPTION

The 'geteuid' system call returns the 16 bit effective user id of the user who executed the current task. The effective user id may be different than the real user id if the task that is executing has set-user-id permissions.

CALLING SEQUENCE

int geteuid() returns user id of task

SEE ALSO

setuid (sys)
getuid (sys)

RETURN VALUES

The effective user id is returned.

NOTES

This is actually implemented as a call to the 'getuid (sys)' system call.

NAME

getgid - get group id of task

DESCRIPTION

The 'getgid' system call returns the 16 bit real group identification number of the current task. The real group id may be different from the effective group id if the task has the set-group-id permission bit set.

CALLING SEQUENCE

```
int getgid()            returns group id of task
```

ASSEMBLY INTERFACE

```
move.l  #23,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

getegid (sys)

RETURN VALUES

The real group id is returned.

NAME

getpgrp - get the process group id of a task

DESCRIPTION

The 'getpgrp' system call is used to get the process group id of the currently running task. Process groups are used to determine which tasks are to recieve certain signals normally generated by terminals (i.e., INT and QUIT).

CALLING SEQUENCE

```
int getpgrp()            returns process group id
```

ASSEMBLY INTERFACE

```
move.l  #44,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

signal (sys)

RETURN VALUES

The process group id is returned.

NAME
getpid - get current task process id

DESCRIPTION
The 'getpid' system call returns the process id of the current task.

CALLING SEQUENCE
```
int getpid()            returns task id
```

ASSEMBLY INTERFACE
```
move.l  #20,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO
maketask (sys)
fork (sys)
exec (sys)

RETURN VALUES
The process id of the task is returned.

NAME
getuid - get user id of task

DESCRIPTION
The 'getuid' system call returns the 16 bit real user id of the user who executed the current task. The real user id may be different than the effective user id if the task that is executing has set-user-id permissions.

CALLING SEQUENCE
```
int getuid()            returns user id of task
```

ASSEMBLY INTERFACE
```
move.l  #24,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO
setuid (sys)
getgid (sys)

RETURN VALUES
The real user id is returned.

NAME

gtty - get terminal control information

DESCRIPTION

This returns the terminals current input/output configuration in the buffer provided. See 'stty (sys)' for a detailed explanation of the tty buffer.

CALLING SEQUENCE

```
#include <sgtty.h>
int gtty(fd,ttybuf)        returns -1 for error, 0 otherwise
int fd;                    file descriptor of char special file
```

ASSEMBLY INTERFACE

```
move.l  #ttybuf,-(sp)   ;move address of ttybuf onto stack
move    fd,-(sp)        ;push fd
move.l  sp,r8           ;current sp points to arg block
move.l  #36,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

stty (sys)

RETURN VALUES

A negative number is returned if the file descriptor does not refer to an open terminal device. Zero is returned if there was no error.

NAME

ioctl - device specific control commands

DESCRIPTION

The 'ioctl' system call is for general purpose device control. The arguments to 'ioctl' are as follows: The first is a file descriptor such as is returned by the 'open (sys)' or 'creat (sys)' system calls. Next is the 'ioctl' command. The third argument is a pointer to a buffer area that is used by the command or an argument to the command. For device specific usage of this command use the device documentation under section '(dev)'. In particular, the tty documentation is under 'tty (dev)'.

CALLING SEQUENCE

```
#include <sys/ioctl.h>
int ioctl(fd, cmnd, argp)
int fd;                         //file descriptor
int cmnd;                       //Command
struct ... *argp;               //pointer to buffer
```

ASSEMBLY INTERFACE

```
move.l  argp,-(sp)              ;push pointer to buffer
move    cmnd,-(sp)              ;push the command
move    fd,-(sp)                ;push the file descriptor
move.l  sp,r8                   ;pass the pointer to arguments
move    #51,r0                  ;system call number
trap    #0                      ;trap to system
```

SEE ALSO

stty (sys)
gtty (sys)
open (sys)
creat (sys)
tty (dev)
floppy (dev)

RETURN VALUES

-1 is returned if there is any error in doing the command, 0 otherwise.

NAME

kill - kill a process

DESCRIPTION

'kill' will stop the processes with the given process id. It sends the specified signal to the specified task. Typically signal 9 (uncatchable kill) or 15 (catchable kill) are used. A system signal may be sent to another task only if: the effective user id of the sending task is the same as the real user id of the recieving task, the effective user id of the sending task is the super user, or the task is sending a signal to itself. A user signal may be sent from any task to any task, however, the default action for a user signal is to ignore it.

If the process id is a positive number, the signal is sent to that specific process, including process 1, the init process. If the process id is zero, the signal will be sent to all processes whose process group id is the same as the sending processes process group id. If the process id is -1, the signal will be sent to all processes whose real user id is the same as the sending processes effective user id. If the process id is -1 and the effective user id of the sending process is the super user, the signal will be sent to all processes. If the process id is negative, but not -1, the signal will be sent to all processes whose process group id is the absolute value of the process id. Process 0, the swapper process, will never be sent a signal. Process 1, the init process, may only be sent a signal explicitly.

Note: kill is the same as 'ssignal (sys)'.

CALLING SEQUENCE

```
int kill(pid,signo)  returns -1 for error, 0 otherwise
int pid;                process id to send signal to
int signo;              signal number to send
```

ASSEMBLY INTERFACE

```
move    signo,-(sp)     ;push signo
move    pid,-(sp)       ;push pid
move.l  sp,r8           ;current sp points to arg block
move.l  #9,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

signal (sys)
ssignal (sys)
kill (cmnd)

RETURN VALUES

Zero is returned if there was no error.

NAME

link - create a new name for a file

DESCRIPTION

The 'link' system call allows a task to create another name for an existing file. An entry with the given file name is made in the target directory with the same index record number as the original file. No actual file data is copied. The file will simply be available with both names. It is not possible to link files across mounted filesystem boundaries.

CALLING SEQUENCE

```
int link(ofn,nfn)       returns -1 if error, 0 otherwise
char *ofn;              existing path name of file
char *nfn;              new path name of file
```

ASSEMBLY INTERFACE

```
move.l  #nfn,-(sp)      ;push address of nfn onto stack
move.l  #ofn,-(sp)      ;push address of ofn onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #26,r0          ;system call number
trap    #0
```

SEE ALSO

unlink (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the new file link could not be made. This is caused by directories being unwritable or cross filesystem links.

NAME

lock - restrict access to an area of a file (REGULUS)

DESCRIPTION

The 'lock' system call allows the user to restrict access to a file or portion of a file. The restrictions can be on reads, writes, locks or combinations of these accesses. In addition, the 'lock' system call can be used to test the locked status of a file.

For flag values of 0, 1, or 2, the requesting task is suspended until the lock can be put into effect. For flag values of 3, 4, or 5, an error return is given to the requesting task if the lock can not be put into effect.

CALLING SEQUENCE

```
lock(fd,lowadd,highadd,flags)    returns -1 if error, 0 otherwise
int fd;                  file descriptor
long lowadd;             low address of file to lock
long highadd;            high address of file to lock
int flags;               locking mode of file, values are:
                         0: 'lock' calls only are restricted
                         1: 'lock' and 'write' calls are restricted
                         2: 'lock', 'write' and 'read' calls are
                            restricted.
                         3: same as 0, error return if
                            couldn't lock.
                         4: same as 1, error return if
                            couldn't lock.
                         5: same as 2, error return if
                            couldn't lock.
```

ASSEMBLY INTERFACE

```
move    flags,-(sp)     ;push flags
move.l  highadd,-(sp)   ;push highadd
move.l  lowadd,-(sp)    ;push lowadd
move    fd,-(sp)        ;push fd
move.l  sp,r8           ;current sp points to arg block
move.l  #39,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

unlock (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the flags indicate error returns are desired and the stated condition is already restricted.

NAME

lread - read >64K bytes from a file

DESCRIPTION

The 'lread' system call attempts to read the specified number of bytes from an open file descriptor into the buffer. The file descriptor can refer to either a file or a device. The read begins at the current location of the file pointer associated with the specified file descriptor. The file pointer may be reset using the 'lseek' system call. A zero will be returned on end of file begin reached, otherwise the number of bytes actually read will be returned.

CALLING SEQUENCE

```
long lread(fd,uadd,nbytes)   returns number of bytes read (0=EOF)
int fd;                      file descriptor
char *uadd;                  user address to read to
long nbytes;                 number of bytes to read
```

ASSEMBLY INTERFACE

```
move    nbytes,-(sp)     ;push nbytes
move.l  #uadd,-(sp)      ;push address of uadd onto stack
move    fd,-(sp)         ;push fd
move.l  sp,r8            ;current sp points to arg block
move.l  #58,r0            ;system call number
trap    #0               ;system call
```

SEE ALSO

open (sys)
write (sys)
pipe (sys)

RETURN VALUES

-1 is returned if there is an error. 0 is returned for end-of-file. The number of bytes actually read is returned if there was no error and you are not at end-of-file.

BUGS

The actual number of bytes read may not be equal to the number of bytes requested. This may or may not be an error. In particular, when reading from a character device (terminal), at most 1 line (up to a carriage return) will be read at a time.

NAME

lsbrk - add 'n' bytes to task break

DESCRIPTION

The 'lsbrk' system call allows the user task to add incremental amounts of bss memory to the end of the task memory space. This call works similarly to the 'brk' system call, except that the value specified is added to the current break pointer.

The amount specified is passed as a long word.

CALLING SEQUENCE

```
char *lsbrk(incr)        returns old break address or -1 for error
long incr;               number of additional bytes to allocate
```

ASSEMBLY INTERFACE

```
move.l #incr,r8
move.l #17,r0
trap   0
```

SEE ALSO

brk (sys)
lsbrk (sys)

RETURN VALUES

The old break address is returned if there were no errors. -1 is returned if it was not possible to increase the break the specified amount.

NOTES

On systems with no memory management, 'lsbrk' has a fixed amount of bss memory allocated at load time. When this is exhausted, 'lsbrk' will always return -1. This amount of memory may be set or changed by 'setstack (cmnd)'.

NAME

lseek - position file pointer

DESCRIPTION

The 'lseek' system call allows the user to position the file pointer for an open file. The file pointer references the place in the file where the next read or write will be performed.

CALLING SEQUENCE

```
long lseek(fd,offset,flag) returns location of next read/write
int fd;                    file descriptor
long offset;               byte offset
int flag;                  flag for calculating file position:
                               0:  bytes from beginning of file
                               1:  bytes from current position
                               2:  bytes from end of file
```

ASSEMBLY INTERFACE

```
move.l  offset,-(sp)    ;push offset
move    fd,-(sp)       ;push fd
move.l  sp,r8          ;current sp points to arg block
move.l  #19,r0         ;system call number
trap    #0             ;system call
```

SEE ALSO

seek (sys)
read (sys)
write (sys)

RETURN VALUES

The file offset where the next read/write will take place will be returned if no error occured. -1 is returned if the file pointer can not be moved. Positioning the file pointer past the end of the file and then writing will result in a block of null bytes filling the empty space.

NAME

lwrite - write >64K bytes to a file

DESCRIPTION

The 'lwrite' system call is used to write a specified number of bytes to an open file descriptor. The specified number of bytes are transfered from the user-supplied buffer to the system file buffers.

CALLING SEQUENCE

```
long lwrite(fd,uadd,nbytes)  returns number of bytes written
int fd;                      file descriptor
char *uadd;                  user address to write from
long nbytes;                 number of bytes to write
```

ASSEMBLY INTERFACE

```
move    nbytes,-(sp)     ;push nbytes
move.l  #uadd,-(sp)      ;push address of uadd onto stack
move    fd,-(sp)         ;push fd
move.l  sp,r8            ;current sp points to arg block
move.l  #59,r0            ;system call number
trap    #0               ;system call
```

SEE ALSO

open (sys)
pipe (sys)
creat (sys)
read (sys)

RETURN VALUES

The number of bytes actually written are returned. It is generally considered an error if this does not equal the number of bytes that were requested to be written. -1 is returned if the file descriptor does not reference an open file or if the file was not opened with writing allowed.

NAME

maketask - make a new task

DESCRIPTION

The 'maketask' system call is the REGULUS general purpose task creation call. The options for the 'maketask' system call allow you to create a new task from a specified object file (fork + exec), create a copy of the calling task (fork), overlay the current task with a specified object file (execv), create a real-time task, or create a "traced" child task.

The priority of the child task may also be set. Priorities may be set less than or equal to the parent task's priority. A priority of -1 will make the child task have the same priority as the parent.

If the flags indicate a FORK, the new task inherits copies of the old task's open files and signal environment, including caught and ignored signals. If the flags indicate an FEXEC or OVERLAY, the new task inherits the standard input, output and error files only, along with the old task's ignored signals.

If an illegal priority is specified then the parent's priority is used. Undefined flag bits are ignored. Note that the fork and execv system calls in C are actually translated into the appropriate 'maketask' call.

CALLING SEQUENCE

```
int maketask(fname,flags,priority,argp,envp)
char *fname;        file name of executable program
int flags;          flags:
                        000: exec object program (FEXEC)
                        001: real-time task      (REAL)
                        002: fork                (FORK)
                        004: overlay (exec)      (OVERLAY)
                        010: set trace bit       (TRACE)
                        020: environment present (ENPRES)
int priority;       priority of task (<= pri of parent)
char *argptrs[];    block of argument pointers, each ended
                    with a null and the block ended with NULL.
char *envp[];       block of environment pointers, each ended
                    with a null and the block ended with NULL.
```

ASSEMBLY INTERFACE

```
move.l  #envp,-(sp)     ;environment block address
move.l  #argptrs,-(sp)  ;argument block address
move    priority,-(sp)  ;push priority
move    flags,-(sp)     ;push flags
move.l  #fname,-(sp)    ;push fname address onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #2,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

fork (sys)
exec (sys)

RETURN VALUES

Zero is returned to the child task of a FORK. The process id of the child task is returned to the parent after a FORK or FEXEC. -1 is returned if the new task could not be created.

BUGS

Too large an argument list to an FEXEC will cause 'maketask' to fail. Each argument requires (4 + length of arg string + 1) bytes in a static operating system buffer. If this space is exhausted often, the size of the buffer should be expanded. Current implementations have 1K bytes available.

NAME

mknod - make a special file or directory

DESCRIPTION

The 'mknod' system call is used to make special file entries in the filesystem. Special file entries are those for devices and directories. The mode bits passed to the system call determine whether you are making a special file corresponding to a device or directory. The entry is made with protections that are the specified protections 'anded' with the complement of the current user's umask.

The access mode bits are discussed in the chmod (cmnd) document. The special device mode bits are defined in 'stat.h'.

FIDIR	directory
FICHR	character special file
FIBLK	block special file
FINPIPE	fifo, named pipe

If the mode bits specify a directory, block, or character special file only the super-user may execute this call. If a named pipe (or fifo) file is specified, any user may execute this call.

CALLING SEQUENCE

```
#include <stat.h>

int mknod(fname,mode,info)    returns -1 if error, 0 otherwise
char *fname;                  name of new special file or directory
int mode;                     mode bits for special file
int info;                     primary and secondary device numbers for
                              device special files (ignored for directo
```

ASSEMBLY INTERFACE

```
move    info,-(sp)        ;push info
move    mode,-(sp)        ;push mode
move.l  #fname,-(sp)      ;push address of fname onto stack
move.l  sp,r8             ;current sp points to arg block
move.l  #27,r0            ;system call number
trap    #0                ;system call
```

SEE ALSO

mkdir (cmnd)
stty (sys)
umask (sys)
filesys (files)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the node could not be created.

NAME

mount - mount a device as a filesystem

DESCRIPTION

The 'mount' system call informs the operating system that a filesystem is present on the specified directory. From that point on, all references to the directory are translated into references to the root directory of the mounted filesystem. The previous contents of the directory are unavailable as long as the directory is mounted.

The special file must correspond to a block special device. The directory must be present and not currently in use. If the rwflag is 1, the filesystem is mounted read-only. Physically write-protected devices must be mounted with this flag set.

This call is restricted to the super-user.

CALLING SEQUENCE

```
int mount(special,name,rwflag)     returns -1 if error, 0 otherwise
char *special;              path name to a block special file
char *name;                 path name to directory to mount on
int  rwflag;                read/write flag
```

ASSEMBLY INTERFACE

```
move    rwflag,-(sp)
move.l  #name,-(sp)      ;push address of name onto stack
move.l  #special,-(sp)   ;push address of special onto stack
move.l  sp,r8            ;current sp points to arg block
move.l  #41,r0           ;system call number
trap    #0
```

SEE ALSO

umount (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the directory is in use, the filesystem is currently mounted somewhere else, the special file does not exist, is in use, or is the wrong type, or the user is not the super-user.

NAME

nice - change task priority

DESCRIPTION

The 'nice' system call allows a user to alter the priority of an already running task. The super-user may raise the priority of a given task, other users may only lower the scheduling priority. Higher numbers are higher priority.

The lowest possible task priority is 0, and the highest is 255. Real-time and non real-time tasks are on different scheduling queues, and the real-time queue is always executed before the non real-time queue. This essentially means that real-time priority 0 is greater than non real-time priority 255.

The actual task priority is changed to the sum of the increment given and the current task priority. An increment of 0 will return the current actual task priority.

Because of the design of REGULUS, it is not possible to schedule a task with a priority above the system scheduler.

CALLING SEQUENCE

```
int nice(incr)      returns -1 for error, new priority otherwise
int incr;           increment to add to current priority
```

ASSEMBLY INTERFACE

```
move.l  #incr,r8    ;move address of incr to r8
move.l  #28,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

nice (cmnd)
ps (cmnd)

RETURN VALUES

The new actual task priority is returned if there was no error. -1 is returned if the user is not allowed to change the priority to the resulting value.

NAME

open - open a file for reading and/or writing

DESCRIPTION

The 'open' system call is used to open an already existing file, or create a file dependant upon the argument flag. The flag consists of bit fields which designate the way in which the file is to be opened. The various fields are or'd together. The type of access requested is specified by :

O_RDONLY	Open for read only
O_WRONLY	Open for write only
O_RDWR	Open for reading and writing

Other items which may be specified :

O_CREAT	If the file does not currently exist, create a file who's ownership is that of the process's effective user. The file will be created with the specified mode, modified only by the effective user's UMASK. Regardless of the creation mode, the access allowed is that which is specified by the flag bits for the open.
O_TRUNC	If the file exists and the O_CREAT flag bit is set truncate the file's length to zero.
O_EXCL	Open for exclusive use. If O_CREAT flag bit is also set and the file already exists, this bit will cause the open to fail.
O_APPEND	The file pointer will be reset to the end of the file before each write takes place.
O_NDELAY	Open for no delay. Reads from ttys and fifos will return immediately if no data can be read. Writes to fifos will return immediately if no space is currently available.

The file pointer will be left pointing to the beginning of the file unless the file is opened in append mode. The file descriptor of the opened file will be returned upon a successful open.

CALLING SEQUENCE

```
#include <fcntl.h>

int open(fname,flag[,mode])
char *fname;            pointer to file name
int flag, mode;         flag for type of open, file mode
```

ASSEMBLY INTERFACE

```
move    flag,-(sp)      ;push flag
```

```
move.l  #fname,-(sp)    ;push address of fname onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #5,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

creat (sys)
write (sys)

RETURN VALUES

The file descriptor for the opened file is returned if there was no error. -1 is returned if the file could not be opened due to a protection violation, file not existing or too many files open already.

NAME

pause - put a process to sleep

DESCRIPTION

The 'pause' system call puts the process to sleep until a signal is sent to the process. Signal catching takes place as usual, and 'pause' will return if the signal was caught.

CALLING SEQUENCE

```
int pause()              // No returns
```

ASSEMBLY INTERFACE

```
move.l  #47,r0          ; Pass system call number
trap    #0              ; Call the system
```

SEE ALSO

signal (sys)
alarm (sys)
sleep (subs)

RETURN VALUES

No return value.

NAME

phys - map system address to user address

DESCRIPTION

The 'phys' system call maps the specified system addresses to addresses in the user address space. This allows a user program to access hardware devices directly. Only the super-user may do this call, as it is very dangerous. No checking is done on the address for validity or pointing to a 'sensitive' kernel area. The maximum phys map length depends on the MMU hardware and the system configuration, typically you can expect no more than 4K bytes.

This works somewhat differently than Version 7 phys, as the task is not locked into memory. Unfortunately, this uses the COMMON segment to map addresses. Therefore, you cannot have both a shared common and a phys active at the same time. To break the phys connection, you can use the 'freecomm (sys)' system call.

CALLING SEQUENCE

```
char *phys(sadd,length)  returns address or -1 for error
char *sadd;              system address to map
long length;             length of address to map in bytes
```

ASSEMBLY INTERFACE

```
move.l  length,-(sp)     ;push length of common area
move.l  sadd,-(sp)       ;push system address
move.l  sp,r8            ;current sp points to arg block
move.l  #57,r0           ;system call number
trap    #0               ;system call
```

SEE ALSO

freecomm (sys)

RETURN VALUES

-1 is returned if there is an error and errno is set. Phys returns the physical address mapped to a user virtual address. Note that the return address will probably not equal sadd.

NAME

pipe - create a pipeline

DESCRIPTION

The 'pipe' system call returns 2 file descriptors to be used as the input and output ends of a pipeline. I/O through pipelines are buffered in the system swap space.

CALLING SEQUENCE

```
pipe(fds)                 returns -1 for error, 0 otherwise
int fds[2];               file descriptors,fd[0]=read,fd[1]=write
```

ASSEMBLY INTERFACE

```
move.l  #fds,-(sp)        ;push address of fds onto stack
move.l  sp,r8             ;current sp points to address of fds
move.l  #37,r0            ;system call number
trap    #0                ;system call
```

SEE ALSO

dup (sys)
close (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if there were too many open files.

BUGS

The task is responsible for setting up the tasks to read and write the pipeline.

NAME

ptrace - trace a child task for debugging

DESCRIPTION

The 'ptrace' system call is a method for controlling a child process for debugging purposes. Upon reception of a signal the child process stops. The parent process is notified of the signal's occurance and can then determine what should happen to the child process. The 'ptrace' system call is implemented as a 'trace (sys)' call.

CALLING SEQUENCE

```
int ptrace(request,pid,cadd,value)
int request;            request id
                          0 - set child to traced
                          1 - read child memory
                          2 - read child's 'D' space
                          3 - read child's s_tasktab
                          4 - write child memory
                          5 - write child's 'D' space
                          6 - write child's s_tasktab
                          7 - send signal to child
                          8 - kill child
int pid;                process id of traced child
char *cadd;             child address to read/write word
int value;              value to write to child space
```

ASSEMBLY INTERFACE

```
move.l  #cadd,-(sp)     ;push address of cadd onto stack
move.l  #padd,-(sp)     ;push address of padd onto stack
move    nbytes,-(sp)    ;push nbytes
move    pid,-(sp)       ;push pid
move    request,-(sp)   ;push request
move.l  sp,r8           ;current sp points to arg block
move.l  #33,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

maketask (sys)
trace (sys)

RETURN VALUES

Zero is returned if there was no error, or -1 for error. Errno is set if there was an error.

NAME

read - read bytes from file

DESCRIPTION

The 'read' system call attempts to read the specified number of bytes from an open file descriptor into the buffer. The file descriptor can refer to either a file or a device. The read begins at the current location of the file pointer associated with the specified file descriptor. The file pointer may be reset using the 'lseek' system call. A zero will be returned on end of file begin reached, otherwise the number of bytes actually read will be returned.

CALLING SEQUENCE

```
int read(fd,uadd,nbytes)    returns number of bytes read (0=EOF)
int fd;                     file descriptor
char *uadd;                 user address to read to
int nbytes;                 number of bytes to read
```

ASSEMBLY INTERFACE

```
move    nbytes,-(sp)    ;push nbytes
move.l  #uadd,-(sp)     ;push address of uadd onto stack
move    fd,-(sp)        ;push fd
move.l  sp,r8           ;current sp points to arg block
move.l  #3,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

lread (sys) - for read > 64K
open (sys)
write (sys)
pipe (sys)

RETURN VALUES

-1 is returned if there is an error. 0 is returned for end-of-file. The number of bytes actually read is returned if there was no error and you are not at end-of-file.

NOTES

The actual number of bytes read may not be equal to the number of bytes requested. This may or may not be an error. In particular, when reading from a character device (terminal), at most 1 line (up to a carriage return) will be read at a time.

NAME

sbrk - add 'n' bytes to task break

DESCRIPTION

The 'sbrk' subroutine call allows the user task to add incremental amounts of bss memory to the end of the task memory space. This call works similarly to the 'brk' system call, except that the value specified is added to the current break pointer.

CALLING SEQUENCE

```
char *sbrk(incr)        returns old break address or -1 for erro
int incr;               number of additional bytes to allocate
```

SEE ALSO

brk (sys)
lsbrk (sys)

RETURN VALUES

The old break address is returned if there were no errors. -1 is returned if it was not possible to increase the break the specified amount.

NOTES

This subroutine is provided for Version 6 compatability and is implemented as a call to 'lsbrk (sys)'.

NAME

seek - position file pointer

DESCRIPTION

The 'seek' subroutine call allows the user to position the file pointer for an open file. The file pointer references the place in the file where the next read or write will be performed.

CALLING SEQUENCE

```
int seek(fd,offset,flag)  returns -1 for error, 0 otherwise
int fd;                   file descriptor
int offset;               byte offset
int flag;                 flag for calculating file position:
                          0:   bytes from beginning of file
                          1:   bytes from current position
                          2:   bytes from end of file
                          3:   blocks from beginning of file
                          4:   blocks from current position
                          5:   blocks from end of file
```

SEE ALSO

lseek (sys)

RETURN VALUES

-1 is returned if the file pointer can not be positioned. Zero is returned if there was no error.

NOTES

This call is provided for compatibility with Version 6 programs only. It is actually implemented as an 'lseek (sys)' call.

NAME

setgid - set group id of task

DESCRIPTION

The 'setgid' system call sets the group id of a task to the specified group id. Only the super-user may change the group id of a task.

CALLING SEQUENCE

```
int setgid(groupid)     returns -1 for error, 0 otherwise
int groupid;            new group id for task
```

ASSEMBLY INTERFACE

```
move.l  #groupid,r8     ;move address of groupid to r8
move.l  #29,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

setuid (sys)
getgid (sys)
getuid (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the user is not the super-user.

BUGS

Nonsense values for the group id are allowed.

NAME

setpgrp - set the process group of a task

DESCRIPTION

The 'setpgrp' system call is used to change the process group id of the currently running task. Process groups are used to determine which tasks are to receive certain signals normally generated by terminals (i.e., INT and QUIT). The process group id is made to be the process id of the calling process. This allows the system to determine the process group leader.

CALLING SEQUENCE

```
int setpgrp()              returns new process group id
```

ASSEMBLY INTERFACE

```
move.l  #43,r0             ;system call number
trap    #0                 ;system call
```

SEE ALSO

signal (sys)

RETURN VALUES

The new process group id is returned.

NAME

setuid - set user id of task

DESCRIPTION

The 'setuid' system call sets the user id of the current task to the specified user id. Only the super-user may execute this system call.

CALLING SEQUENCE

```
int setuid(userid)      returns -1 for error, 0 otherwise
int userid;             new user id for task
```

ASSEMBLY INTERFACE

```
move.l  #userid,r8      ;move address of userid to r8
move.l  #30,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

setgid (sys)
getuid (sys)
getgid (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the user is not the super-user.

BUGS

Nonsense values for the user id are allowed and will be set.

NAME

signal - catch or ignore signals

DESCRIPTION

The 'signal' system call allows a user task to handle an asynchronous event.

The 'signal' system call allows for the setting of a response to another task doing an ssignal (kill) system call, or the operating system detecting an "unusual" event, such as divide check. Signals numbered 0-31 are special system defined signals and are handled in the following way:

If no 'signal' has been done for that signal number the default signal action is taken. The default signal action is to terminate the task and in some cases produce a core dump.

If 'signal' is called with an even address, then that address is taken to be the entry point of a signal catching routine. The process is then interrupted when the signal occurs and the signal catching routine is called. When the signal catching routine returns the process continues execution from the point at which it was interrupted, unless the process stack has been reset using the stack manipulation routines 'setstack (subs)' and 'reset (subs)'.

If 'signal' is called with an odd address, the signal is ignored.

Signals numbered 32-63 are user-definable signals. The default action for user signals is to ignore them. Hence, you can either catch user signals or ignore them. Also, user signals can be sent to any task without regard for task ownership.

Note that "signal" is a C language routine which maintains a table of addresses of the user specified routines. It is highly recomended that this routine be used instead of the system call 'makestab (sys)' due to a number of serious timing considerations.

CALLING SEQUENCE

```
char *signal(signo,sigadd)    returns -1 for error, old signal address
int signo;                    signal number for processing
int *sigadd;                  signal catching routine address
                                  0:      restore default action
                                  odd:    ignore signal
                                  even:   user signal catching
```

SEE ALSO

makestab (sys), ssignal (sys)

FILES

/usr/include/signal.h include file of signal defines

RETURN VALUES
The previous value of the signal is returned.

System Signal Summary

The following signals are currently defined for REGULUS. If the signal number is followed by an asterisk, then that signal will cause a core dump if not ignored. The names in capital letters are the defines currently used in the include file 'signal.h'.

1	SIG_HUP	Hangup
2	SIG_INT	Interrupt
3*	SIG_QUIT	Quit
4*	SIG_ILL	Illegal Instruction
5*	SIG_TRACE	Trace Trap
6	SIG_IOT	IOT on pdp-11
7	SIG_EMT	EMT on pdp-11
8*	SIG_FLT	Floating Point Exception
9	SIG_KILL	Kill
10*	SIG_BUS	Bus Error
11*	SIG_SEG	Segmentation Violation
12*	SIG_SYS	Bad Argument to System Call
13	SIG_PIPE	Write on pipe with no one to read
14	SIG_ALRM	Alarm clock
15	SIG_TERM	Terminate (catchable kill)
16	SIG_USR1	User 1
17	SIG_USR2	User 2
18	SIG_CHLD	Death of Child (not implemented)
19	SIG_PWR	Power Failure (not implemented)
20*	SIG_ADD	Address error
21*	SIG_DIV	Zero divide
22*	SIG_CHK	Chk instruction
23*	SIG_DVR	Trapv instruction
24*	SIG_PRIV	Privilege violation
25*	SIG_U1T	1010 instruction trap
26*	SIG_U2T	1111 instruction trap
27*	SIG_INV	Invalid trap
28*	SIG_BRK	Breakpoint trap
29*	SIG_MEM	Out of memory
30*	SIG_REL	Bad relocation bits
31*	SIG_UNDER	User stack underflow

The following defines are also available:

0	SIG_DFL	restore default signal action
-1	SIG_IGN	ignore signal

Note that signal number 9 is a special KILL signal, which can neither be caught or ignored.

NAME

ssignal - send signal to task (REGULUS)

DESCRIPTION

The 'ssignal' system allows a task to send a signal to another task. Valid signal numbers are 0 through 63. Signals 0 through 31 are System defined signals, and correspond to (typically hardware) specific events. The signal 0 is a special signal which is never sent, but can be used by ssignal to check the validity of the receiving process. Signals 32 through 63 are User defined signals, which have no special meaning to the system. A system signal may be sent to another task only if: the effective user id of the sending task is the same as the real user id of the recieving task, the effective user id of the sending task is the super user, or the task is sending a signal to itself. A user signal may be sent from any task to any task, however, the default action for a user signal is to ignore it.

If the process id is a positive number, the signal is sent to that specific process, including process 1, the init process. If the process id is zero, the signal will be sent to all processes whose process group id is the same as the sending processes process group id. If the process id is -1, the signal will be sent to all processes whose real user id is the same as the sending processes effective user id. If the process id is -1 and the effective user id of the sending process is the super user, the signal will be sent to all processes. If the process id is negative, but not -1, the signal will be sent to all processes whose process group id is the absolute value of the process id. Process 0, the swapper process, will never be sent a signal. Process 1, the init process, may only be sent a signal explicitly.

Note: ssignal is the same as 'kill (sys)'.

CALLING SEQUENCE

```
int ssignal(pid,signo)  returns -1 for error, 0 otherwise
int pid;                process id to send signal to
int signo;              signal number to send
```

ASSEMBLY INTERFACE

```
move    signo,-(sp)     ;push signo
move    pid,-(sp)       ;push pid
move.l  sp,r8           ;current sp points to arg block
move.l  #9,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

signal (sys)
kill (sys)

kill (cmnd)

RETURN VALUES
Zero is returned if there was no error.

NAME

stat - get file status by file name

DESCRIPTION

The 'stat' system call is used to get file information about the file whose path name is specified. If there are no errors, the index record number of the file is returned. The information in the given structure is available.

CALLING SEQUENCE

```
long stat(fn,buf)       returns -1 for error, index block number
char *fn;                file name of file to stat

struct stat {           address of buffer to place status informat:
    int     i_magic;        /* magic number to id indirect blocks
    long    st_size;        /* file size */
    int     st_uid;         /* file owner id */
    char    st_gid;         /* group id of file */
    char    st_nlink;       /* file reference count */
    long    st_mtime;       /* file modification date and time */
    long    st_atime;       /* file access date and time */
    int     st_mode;        /* file type and access mode */
    int     i_nul;          /* pad for word */
    int     st_rdev;        /* device for character and block dev
    long    i_nu2;          /* pad for block */
    int     i_nu3;          /* pad for word */
    int     st_dev;         /*  device number where file resides
} *buf;
```

The above structure resides in the '#include' file <stat.h>.

ASSEMBLY INTERFACE

```
move.l  #uadd,-(sp)     ;push address of uadd onto stack
move.l  #fn,-(sp)       ;push address of file name
mov.l   sp,r8           ;current sp points to arg block
move.l  #21,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

fstat (sys)

RETURN VALUES

The index record number is returned if there was no error. -1 is returned if the named file does not exist or if any of the directories leading to it are unreadable due to protections.

NAME

stime - set time of day

DESCRIPTION

The 'stime' system call is used to set the REGULUS time-of-day clock. The 'stime' system call is restricted to the superuser.

CALLING SEQUENCE

```
int stime(tadd)          returns -1 for error, 0 otherwise
long *tadd;              address of long word for new time of day
```

ASSEMBLY INTERFACE

```
move.l    #tadd,-(sp)    ;push address of tadd onto stack
move.l    sp,r8          ;current sp points to tadd address
move.l    #25,r0         ;system call number
trap      #0             ;system call
```

SEE ALSO

time (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if you are not the super-user.

NOTES

No check on the validity of the new time is performed. Times are assumed to be the number of seconds since Midnight Jan. 1, 1970.

NAME

stty - set terminal control information

DESCRIPTION

The 'stty' system call allows the user to change a terminal's current input/output configuration. This has been replaced by the ioctl system call, however interface routines exist for compatability with Version 6 and Version 7 Unix.

The tty buffer has the following structure:

```
struct tty {
    char t_ispeed; input baud rate (see below)
    char t_ospeed; output baud rate
    char t_erase; erase character
    char t_kill; kill character
    int t_modes; bit masks for modes };
```

The speed bytes change the speed of the terminal if the device is capable of changes. Changing the speed on a device not so equipped or to a speed that is not implemented for that device is useless. The following table shows the correspondence between the speed settings and the actual baud rates implemented:

B0	0	0 baud (hang up)
B50	1	50 baud
B75	2	75 baud
B100	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	19200 baud
EXTB	15	38400 baud

The terminal mode word sets the way REGULUS handles certain terminal dependent operations, including character mapping, echo, tabs, etc. The bit masks and their meanings are as follows:

ALLDELAY	0177400	All delay settings bits
BSDELAY	0100000	Backspace Delay
BS0	0000000	No backspace delay (default)
BS1	0100000	Unimplemented
VTDELAY	0040000	Form feed delay
FF0	0000000	No form feed delay
FF1	0040000	Delay approximately .10 sec.

CRDELAY	0030000	Carriage return delay
CR0	0000000	No carriage return delay
CR1	0010000	Delay approximately .10 sec.
CR2	0020000	Delay approximately .20 sec.
CR3	0030000	Delay approximately .30 sec.
TBDELAY	0006000	Tab delay
TAB0	0000000	No tab delay
TAB1	0002000	Delay approximately .10 sec.
TAB2	0004000	Delay approximately .20 sec.
XTABS	0006000	Translate tabs to spaces
NLDELAY	0001400	New line (line feed) delay
NL1	0000400	Delay approximately .10 sec.
NL2	0001000	Delay approximately .20 sec.
NL3	0001400	Delay approximately .30 sec.
EVENP	0000200	Even parity
ODDP	0000100	Odd parity
RAW	0000040	Raw mode, wake up on all characters
CRMOD	0000020	map LF to CR-LF, echo CR or LF as CR-LF
ECHO	0000010	Echo input
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Same as raw, but handle INTR and QUIT
TANDEM	0000001	Enable XON-XOFF control

Delay bits specify one of a number of delays. A value of zero is used for no delay. All other values are system dependent.

Raw mode indicates that each character is to be passed to the reading process as soon as it is available. No special handling of the end-of-file (EOT), interrupt (DEL), or backspace is done. The line editor must be disabled with raw mode.

Cbreak mode is like raw mode on input in that the process is awakened on each character, but it also does kill and interrupt processing. No erase or input mapping is done. All output processing is still done. The cbreak mode is incompatible with raw mode and the line editor, so only one of the three should be used at any one time.

REGULUS will map carriage returns and line feeds into a carriage return/linefeed pair when enabled.

REGULUS will map uppercase characters into their lowercase equivalents if needed for older terminals.

The tab to blanks mapping bit is required to use other than hardware specified tab sizes.

CALLING SEQUENCE

```
#include <sgtty.h>
int stty(fd,ttybuf)     returns -1 for error, 0 otherwise
int fd;                 file descriptor of char special file
struct tty *ttybuf;     pointer to buffer to tty parameters
```

ASSEMBLY INTERFACE

```
move.l  #ttybuf,-(sp)   ;move address of ttybuf onto stack
move    fd,-(sp)        ;push fd
move.l  sp,r8           ;current sp points to arg block
move.l  #34,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

gtty (sys)

RETURN VALUES

Zero is returned if there were no errors. -1 is returned if the file descriptor does not refer to an open terminal device or the user does not have write permission on that device.

NAME
sync - update system buffers

DESCRIPTION
The 'sync' system call causes all pending disk I/O in the in-core system buffers to be written to disk.

CALLING SEQUENCE
```
int sync()              returns -1 for error, 0 otherwise
```

ASSEMBLY INTERFACE
```
move.l  #32,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO
update (cmnd)

RETURN VALUES
Zero is returned if there was no error.

NAME

sysinfo - get system information (REGULUS)

DESCRIPTION

The 'sysinfo' system call returns specified information about the currently executing REGULUS kernel. This typically saves a tedious lookup through the '/regulus' file for addresses, and also insures that you are obtaining the addresses from the currently running system, which is not necessarily '/regulus'. The information that may be obtained is:

I_MMUSTAT	memory management status, 0=>No MMU, 1=>MMU
I_ROOTDEV	kernel root device
I_NTASKS	number of in-core task table slots
I_INTASKP	starting address of task table
I_SYSNAME	REGULUS system name
I_NODENAME	REGULUS node name
I_RELEASE	REGULUS release string
I_VERSION	REGULUS version string

CALLING SEQUENCE

```
#include <sys/sysinfo.h>
int sysinfo(flag,addr)  returns -1 for error, 0 for OK
int flag;               flag for info to return
int *addr;              pointer to buffer to place info
```

ASSEMBLY INTERFACE

```
move.l  #addr,-(sp)     ;push address of addr onto the stack
move    flag,-(sp)      ;push flag
move.l  sp,r8           ;current sp points to the arg block
move.l  #50,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

mem (dev)

RETURN VALUES

-1 is returned if the requested information flag is invalid, or the buffer address is invalid. 0 is returned otherwise.

NAME
time - get time of day

DESCRIPTION
The 'time' system call returns the system's idea of the time of day. The time is represented as the number of seconds since Midnight January 1, 1970.

CALLING SEQUENCE

```
int time(tadd)          returns -1 for error, 0 otherwise
long *tadd;             address of long word to place time
```

ASSEMBLY INTERFACE

```
move.l    #tadd,-(sp)     ;push address of tadd onto stack
move.l    sp,r8           ;current sp points to tadd address
move.l    #13,r0          ;system call number
trap      #0              ;system call
```

SEE ALSO
stime (sys)

RETURN VALUES
The system idea of the time of day is returned.

NAME

times - return execution times of task

DESCRIPTION

The 'times' system call is used to calculate the execution times of a task and its sub-tasks (children). All times are in 1/10's of a second, and are probably accurate to one second. The children's time is the sum of the task's currently terminated child tasks.

CALLING SEQUENCE

```
int times(tbuf)          returns -1 for error, 0 otherwise
long tbuf[4];            buffer in user space, contents are:
                             tbuf[0]: task user time
                             tbuf[1]: task system time
                             tbuf[2]: children user time
                             tbuf[3]: children system time
```

ASSEMBLY INTERFACE

```
move.l    #tbuf,-(sp)    ;push address of tbuf onto stack
move.l    sp,r8          ;current sp points to tbuf address
move.l    #38,r0         ;system call number
trap      #0             ;system call
```

SEE ALSO

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the buffer address is invalid.

NAME

trace - trace a child task for debugging (REGULUS)

DESCRIPTION

The 'trace' system call allows for the dynamic debugging of tasks. The child task being traced (debugged) must place itself into the "traced" state or be created with the maketask TRACE flag. The parent task may then access the child task's memory and control the execution of the child task. If the child task receives a signal, the signal is first sent to the parent task which can decide whether to allow the signal to be passed to the child or not.

CALLING SEQUENCE

```
int trace(request,pid,nbytes,padd,cadd)
returns -1 for error,
                          0 otherwise.
int request;              request:
   0: set child to "traced"
   1: read child memory
   2: read child D space for separate I & D systems
   3: read child s_tasktab
   4: write child memory
   5: write child D space for separate I & D systems
   6: write child s_tasktab
   7: send signal to child
   8: kill child
int pid;                  process id of traced child
int nbytes;               number of bytes to read/write
char *padd;               parent address to read/write
char *cadd;               child address to read/write
```

ASSEMBLY INTERFACE

```
move.l  #cadd,-(sp)     ;push address of cadd onto stack
move.l  #padd,-(sp)     ;push address of padd onto stack
move    nbytes,-(sp)    ;push nbytes
move    pid,-(sp)       ;push pid
move    request,-(sp)   ;push request
move.l  sp,r8           ;current sp points to arg block
move.l  #33,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

maketask (sys)
ptrace (sys)

RETURN VALUES

Zero is returned if there was no error. A -1 is returned if an error occurred.

NAME

umask - set file creation access mask

DESCRIPTION

The 'umask' system call sets the mask that is used to determine the file access bits of files that are created by the 'creat (sys)' or 'mknod (sys)' system calls. The actual value of the file access bits will be the logical 'and' of the complement of the current mask and the file creation mode given in the system calls 'creat' and 'mknod'.

Each child process inherits the mask value of its parent process.

CALLING SEQUENCE

```
int umask(cmask)        returns previous mask value
int cmask;              new mask value
```

ASSEMBLY INTERFACE

```
move    cmask,(sp)      ;push the new mask
move.l  sp,r8           ;current sp points to arg block
move.l  #45,r0          ;system call number
trap    #0
```

SEE ALSO

creat (sys), mknod (sys)

RETURN VALUES

The old mask value is returned.

NAME

umount - unmount a filesystem from a device

DESCRIPTION

The 'umount' system call is used to remove the correspondence between a directory and a device. The special file must correspond to a block special device which has been previously mounted (see 'mount (sys)'). References to the directory will no longer be mapped to the root directory on the device, and the original files in the directory will again be available.

This call is restricted to the super-user.

CALLING SEQUENCE

```
int umount(special)          returns -1 if error, 0 otherwise
char *special;             path name to a block special file
```

ASSEMBLY INTERFACE

```
move.l  #special,-(sp)  ;push address of special onto stack
move.l  sp,r8           ;current sp points to arg block
move.l  #42,r0          ;system call number
trap    #0
```

SEE ALSO

mount (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the special file does not exist or if the special file was not mounted.

NAME

unlink - remove a filename from a filesystem

DESCRIPTION

The 'unlink' system call clears the directory entry for the specified filename. If this is the last name pointing to the file data, the data blocks for the file are freed.

CALLING SEQUENCE

```
int unlink(fname)       returns -1 if error, 0 otherwise
char *fname;            path name to unlink
```

ASSEMBLY INTERFACE

```
move.l  #fname,-(sp)    ;move address of fname onto stack
move.l  sp,r8           ;current sp points to address of fname
move.l  #31,r0          ;system call number
trap    #0              ;system call
```

SEE ALSO

link (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the file does not exist or if you are not the owner or super-user.

NAME

unlock - remove access restrictions from file (REGULUS)

DESCRIPTION

The 'unlock' system call removes all previously placed restrictions from a file.

CALLING SEQUENCE

```
unlock(fd,lowadd,highadd)    returns -1 if error, 0 otherwise
int fd;                      file descriptor to unlock
long lowadd;                 low address of locked area
long highadd;                high address of locked area
```

ASSEMBLY INTERFACE

```
move.l  highadd,-(sp)    ;push highadd
move.l  lowadd,-(sp)     ;push lowadd
move    fd               ;push fd
move.l  sp,r8            ;current sp points to arg block
move.l  #40,r0           ;system call number
trap    #0               ;system call
```

SEE ALSO

lock (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the file descriptor points to a file that was not previously locked.

NAME

utime - change file accessed, modified times

DESCRIPTION

The 'utime' system call is used to change the last access and modified times associated with a file. The first argument is the name of the file to change. The second argument is a pointer to an array of 2 long numbers. The first long number is used as the file's new access time, and the second long number as the file's modified time.

If the pointer is zero, then the times of the file are changed to be the current system time.

CALLING SEQUENCE

```
utime(fname, times)
char *fname;            //name of file to use
long times[2];          //New accessed and modified times
```

ASSEMBLY INTERFACE

```
move.l times,-(sp)      ;push the address of the 2 times
move.l fname,-(sp)      ;push the file name
move.l sp,r8            ;current sp points to arguments
move.l #46,r0           ;system call number
trap #0                 ;call system
```

SEE ALSO

RETURN VALUES

-1 is returned if there was an error, 0 otherwise.

NAME

wait - wait for a child task to terminate

DESCRIPTION

The 'wait' system call allows a task to suspend its execution until one of its child tasks terminates. If there are no children, 'wait' returns with an error condition. If there are children, the task is suspended until one of the child tasks terminates or a signal is received by the task. The 'wait' call returns the process id of the terminated task.

The parameter to 'wait' is the address of an integer. 'wait' returns the exit status of the child in this location. The exit status is in the form of two 8 bit items. The high byte contains the low byte of the child's exit status and the low byte contains the child's termination status. If there is more than 1 child task, a separate wait for each child is required.

CALLING SEQUENCE

```
int wait(addr)
int *addr;              // address to place child exit status
```

ASSEMBLY INTERFACE

```
move.l  #addr,-(sp)     ;push address of addr onto stack
move.l  sp,r8           ;current sp points to addr address
move.l  #7,r0           ;system call number
trap    #0              ;system call
```

SEE ALSO

maketask (sys)

RETURN VALUES

-1 is returned if there are no children to wait for. The process id of the first terminated child task is returned if there is no error.

NAME

write - write to a file

DESCRIPTION

The 'write' system call is used to write a specified number of bytes to an open file descriptor. The specified number of bytes are transfered from the user-supplied buffer to the system file buffers.

CALLING SEQUENCE

```
int write(fd,uadd,nbytes)  returns number of bytes written
int fd;                    file descriptor
char *uadd;                user address to write from
int nbytes;                number of bytes to write
```

ASSEMBLY INTERFACE

```
move    nbytes,-(sp)     ;push nbytes
move.l  #uadd,-(sp)      ;push address of uadd onto stack
move    fd,-(sp)         ;push fd
move.l  sp,r8            ;current sp points to arg block
move.l  #4,r0            ;system call number
trap    #0               ;system call
```

SEE ALSO

open (sys)
pipe (sys)
creat (sys)
read (sys)

RETURN VALUES

The number of bytes actually written are returned. It is generally considered an error if this does not equal the number of bytes that were requested to be written. -1 is returned if the file descriptor does not reference an open file or if the file was not opened with writing allowed.

NAME

chown - Version 6 change owner of a file

DESCRIPTION

The 'chown' system call is used to change the owner of a file. Only the super-user may execute this call. The numerical ownerid and groupid of the named file is changed to the specified ids. Care should be taken that the new id has been assigned to a user. This is implemented as calls to 'chown (sys)' and 'chgrp (sys)'.

CALLING SEQUENCE

```
int chown(fname,uid)        returns -1 for error, 0 otherwise
char *fname;                name of file to change owner
int uid;                    new owner id of file in low-byte,
                            new group id in high-byte
```

SEE ALSO

chown (sys)
chgrp (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if you are not the super-user.

NAME

fstat - Version 6 get file status by file descriptor

DESCRIPTION

The 'stat' system call is used to get file information about the file whose open descriptor is specified. This is implemented as a function that does a REGULUS 'fstat (sys)' system call, then rearranges the returned values.

Note that the inumber is the REGULUS index record number, truncated to 16 bits, and the size0 is the upper word of the REGULUS file size truncated to 8 bits.

Also note that Version 6 did not have a standard include file with standard definitions.

CALLING SEQUENCE

```
#include <inode.h>
int fstat(fd,buf)       returns -1 for error, 0 otherwise
int fd;                 file descriptor of file to stat
struct inode *buf;      pointer to stat buffer

/* stat structure for Unix Version 6 */

struct inode {
    char major;           /*major device of inode*/
    char minor;           /*major device of inode*/
    int inumber;          /*inode number*/
    int flags;            /*file type flags*/
    char nlinks;          /*number of links*/
    char uid;             /*user id*/
    char gid;             /*group id*/
    char size0;           /*high byte of file size*/
    int size1;            /*low word of file size*/
    int addr[8];          /*block addresses*/
    int actime[2];        /*access date*/
    int modtime[2];       /*modified date*/
};

#define    S_IFMT     0070000
#define    S_IFDIR    0040000
#define    S_IFCHR    0020000
#define    S_IFBLK    0060000
#define    S_IFREG    0000000
#define    S_ISUID    0004000
#define    S_IGUID    0002000
#define    S_ISVTX    0001000
#define    S_IREAD    0000400
#define    S_IWRITE   0000200
#define    S_IEXEC    0000100
```

SEE ALSO

stat (v6/sys)
fstat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the named file does not exist or if any of the directories leading to it are unreadable due to protections.

NAME

getcsw - Version 6 read console switches

DESCRIPTION

The 'getcsw' system call, which is used to read the CPU console switches, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

gtty - Version 6 get terminal control information

DESCRIPTION

This returns the terminals current input/output configuration in the buffer provided. See 'stty (v6/sys)' for a detailed explanation of the tty buffer. This is implemented as an interface routine to 'ioctl (sys)'.

CALLING SEQUENCE

```
#include <sgtty.h>
int gtty(fd,ttybuf)        returns -1 for error, 0 otherwise
int fd;                    file descriptor of char special file
struct sgtty *ttybuf;      pointer to tty buffer
```

SEE ALSO

stty (v6/sys)
ioctl (sys)

RETURN VALUES

A negative number is returned if the file descriptor does not refer to an open terminal device. Zero is returned if there was no error.

NAME

mknod - Version 6 make a special file or directory

DESCRIPTION

The 'mknod' system call is used to make special file entries in the filesystem. Special file entries are those for devices and directories. The mode bits passed to the system call determine whether you are making a special file corresponding to a device or directory. The entry is made with protections that are the specified protections 'anded' with the complement of the current user's umask. This interface routine alters the Version 6 special file mode bits to match the REGULUS special file mode bits.

The access mode bits are discussed in the chmod (cmnd) document. The special device mode bits are defined in <v6/inode.h>. Note that Version 6 did not have a standard include file with standard definitions.

Only the super-user may execute this call.

CALLING SEQUENCE

```
#include <inode.h>

int mknod(fname,mode,info)     returns -1 if error, 0 otherwise
char *fname;                   name of new special file or directory
int mode;                      mode bits for special file
int info;                      primary and secondary device numbers for
                               device special files (ignored for directc
```

SEE ALSO

mkdir (cmnd)
mknod (cmnd)
umask (sys)
filesys (files)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the node could not be created.

NAME

nice - Version 6 change task priority

DESCRIPTION

The nice Version 6 interface routine attempts to make a reasonable mapping of Version 6 nice values to REGULUS nice values. On REGULUS, priorities are always positive, with larger numbers implying higher priority. On Version 6, negative priorities are for system tasks and the smaller the priority value, the higher the priority.

CALLING SEQUENCE

```
int nice(incr)      returns -1 for error, new priority otherwise
int incr;           increment to add to current priority
```

SEE ALSO

nice (cmnd)
ps (cmnd)
nice (sys)

RETURN VALUES

The new actual task priority is returned if there was no error. -1 is returned if the user is not allowed to change the priority to the resulting value.

NAME

profil - Version 6 profile task execution

DESCRIPTION

The 'profil' system call, which is used to get an execution profile of a task, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

stat - Version 6 get file status by file name

DESCRIPTION

The 'stat' system call is used to get file information about the file whose name is specified. This is implemented as a function that does a REGULUS 'stat (sys)' system call, then rearranges the returned values.

Note that the inumber is the REGULUS index record number, truncated to 16 bits, and the size0 is the upper word of the REGULUS file size truncated to 8 bits. Also note that Version 6 did not have a standard include file with standard definitions.

CALLING SEQUENCE

```
#include <inode.h>
int stat(fname,buf)     returns -1 for error, 0 otherwise
char *fname;            file name of file to stat
struct inode *buf;      pointer to stat buffer

/* stat structure for Unix Version 6 */

struct inode {
    char major;          /*major device of inode*/
    char minor;          /*major device of inode*/
    int inumber;         /*inode number*/
    int flags;           /*file type flags*/
    char nlinks;         /*number of links*/
    char uid;            /*user id*/
    char gid;            /*group id*/
    char size0;          /*high byte of file size*/
    int size1;           /*low word of file size*/
    int addr[8];         /*block addresses*/
    int actime[2];       /*access date*/
    int modtime[2];      /*modified date*/
};

#define    S_IFMT     0070000
#define    S_IFDIR    0040000
#define    S_IFCHR    0020000
#define    S_IFBLK    0060000
#define    S_IFREG    0000000
#define    S_ISUID    0004000
#define    S_IGUID    0002000
#define    S_ISVTX    0001000
#define    S_IREAD    0000400
#define    S_IWRITE   0000200
#define    S_IEXEC    0000100
```

SEE ALSO

fstat (v6/sys)
stat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the named file does not exist or if any of the directories leading to it are unreadable due to protections.

NAME

stty - Version 6 set terminal control information

DESCRIPTION

The 'stty' system call allows the user to change a terminal's current input/output configuration. This has been replaced by the ioctl system call, however interface routines exist for compatability with Version 6. It is highly recommended that programs use the REGULUS 'ioctl (sys)' system call rather than the interface routine.

The interface tty buffer has the following structure:

```
struct tty {
    char t_ispeed;      input baud rate (see below)
    char t_ospeed;      output baud rate
    char t_erase;       erase character
    char t_kill;        kill character
    int t_modes;        bit masks for modes
};
```

The speed bytes change the speed of the terminal if the device is capable of changes. Changing the speed on a device not so equipped or to a speed that is not implemented for that device is useless. The following table shows the correspondence between the speed settings and the actual baud rates implemented:

B0	0	0 baud (hang up)
B50	1	50 baud
B75	2	75 baud
B100	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	19200 baud
EXTB	15	38400 baud

The terminal mode word sets the way Version 6 handles certain terminal dependent operations, including character mapping, echo, tabs, etc. The bit masks and their meanings are as follows:

ALLDELAY	0177400	All delay settings bits
BSDELAY	0100000	Backspace Delay
BS0	0000000	No backspace delay
BS1	0100000	Delay approximately .10 sec.
VTDELAY	0040000	Form feed delay

FF0	0000000	No form feed delay
FF1	0040000	Delay approximately .10 sec.
CRDELAY	0030000	Carriage return delay
CR0	0000000	No carriage return delay
CR1	0010000	Delay approximately .10 sec.
CR2	0020000	Delay approximately .20 sec.
CR3	0030000	Delay approximately .30 sec.
TBDELAY	0006000	Tab delay
TAB0	0000000	No tab delay
TAB1	0002000	Delay approximately .10 sec.
TAB2	0004000	Delay approximately .20 sec.
TAB3	0006000	Not implemented
NLDELAY	0001400	New line (line feed) delay
NL1	0000400	Delay approximately .10 sec.
NL2	0001000	Not implemented
NL3	0001400	Not implemented
EVENP	0000200	Even parity (PARENB+~PARODD)
ODDP	0000100	Odd parity (PARENB+PARODD)
RAW	0000040	Raw mode (~ICANON+~ISIG)
CRMOD	0000020	new-line processing (ICRNL+ONLCR)
ECHO	0000010	Echo input
LCASE	0000004	Upper case processing (IUCLC+OLCUC)
XTABS	0000002	Output tabs as spaces
HUPCL	0000001	Hang up on last close

The above mappings of Version 6 control settings to REGULUS control settings do not always act precisely as the Version 6 settings do. For instance, raw mode on Version 6 disables CRMOD, whereas on REGULUS the modes are independent.

CALLING SEQUENCE

```
#include <sgtty.h>
int stty(fd,ttybuf)     returns -1 for error, 0 otherwise
int fd;                 file descriptor of char special file
struct tty *ttybuf;     pointer to buffer to tty parameters
```

SEE ALSO

stty (cmnd)
ioctl (sys)
gtty (v6/sys)

RETURN VALUES

Zero is returned if there were no errors. -1 is returned if the file descriptor does not refer to an open terminal device or the user does not have write permission on that device.

NAME

times - Version 6 return execution times of task

DESCRIPTION

The 'times' system call is used to calculate the execution times of a task and its sub-tasks (children). All times are in 1/60's of a second, and are probably accurate to one second. The children's time is the sum of the task's currently terminated child tasks. This merely modifies the return values from the 'times (sys)' system call so that the time increments are 1/60's of a second rather than 1/10's of a second.

CALLING SEQUENCE

```
int times(tbuf)          returns -1 for error, 0 otherwise
struct tbuffer *buffer; pointer to times structure returned

struct tbuffer {
    int proc_user_time;
    int proc_system_time;
    int child_user_time[2];
    int child_system_time[2];
};
```

SEE ALSO

times (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the buffer address is invalid.

NAME

acct - Version 7 enable accounting record keeping

DESCRIPTION

The 'acct' system call, which outputs system accounting records on a per-task basis to an accounting file, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

chown - Version 7 change owner of a file

DESCRIPTION

The 'chown' system call is used to change the owner of a file. Only the super-user may execute this call. The numerical ownerid and groupid of the named file is changed to the specified ids. Care should be taken that the new id has been assigned to a user. This is implemented as calls to 'chown (sys)' and 'chgrp (sys)'.

CALLING SEQUENCE

```
int chown(fname,uid,gid)    returns -1 for error, 0 otherwise
char *fname;                name of file to change owner
int uid;                    new owner id of file
int gid;                    new group id of file
```

SEE ALSO

chown (sys)
chgrp (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if you are not the super-user.

NAME

nice - Version 7 change task priority

DESCRIPTION

The nice Version 7 interface routine attempts to make a reasonable mapping of Version 7 nice values to REGULUS nice values. On REGULUS, priorities are always positive, with larger numbers implying higher priority. On Version 7, negative priorities are for system tasks and the smaller the priority value, the higher the priority.

CALLING SEQUENCE

```
int nice(incr)      returns -1 for error, new priority otherwise
int incr;           increment to add to current priority
```

SEE ALSO

nice (cmnd)
ps (cmnd)
nice (sys)

RETURN VALUES

The new actual task priority is returned if there was no error. -1 is returned if the user is not allowed to change the priority to the resulting value.

NAME

profil - Version 7 profile task execution

DESCRIPTION

The 'profil' system call, which is used to get an execution profile of a task, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

stat - Version 7 get file status by file name

DESCRIPTION

The 'stat' system call is used to get file information about the file whose path name is specified. This is implemented as a function that does a REGULUS 'stat (sys)' system call, then rearranges the returned values.

Note that the inode number is the REGULUS index record number truncated to 16 bits and the inode changed time is the REGULUS modified time.

CALLING SEQUENCE

```
#include <stat.h>
int stat(fn,buf)        returns -1 for error, 0 otherwise
char *fn;               file name of file to stat
struct stat *buf;       pointer to stat buffer

/* stat structure for Unix Version 7 */

struct stat {
    int st_dev;         /*device inode is on*/
    int st_ino;         /*inode number*/
    int st_mode;        /*file mode bits*/
    int st_nlink;       /*number of links*/
    int st_uid;         /*owner id*/
    int st_gid;         /*group id*/
    int st_rdev;        /*real device if mount, char or block*/
    long st_size;       /*file size in bytes*/
    long st_atime;      /*access time*/
    long st_mtime;      /*modified time*/
    long st_ctime;      /*inode changed time*/
};

#define     S_IFMT      0170000     /*file type*/
#define     S_IFDIR     0040000     /*directory*/
#define     S_IFCHR     0020000     /*character*/
#define     S_IFBLK     0060000     /*block*/
#define     S_IFREG     0100000     /*regular*/
#define     S_IFMPC     0030000     /*multiplexed char*/
#define     S_IFMPB     0070000     /*multiplexed block*/
#define     S_ISUID     0004000     /*set user id*/
#define     S_IGUID     0002000     /*set group id*/
#define     S_ISVTX     0001000     /*save text*/
#define     S_IREAD     0000400     /*owner read*/
#define     S_IWRITE    0000200     /*owner write*/
#define     S_IEXEC     0000100     /*owner exec*/
```

SEE ALSO

fstat (v7/sys)
stat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if

the named file does not exist or if any of the directories leading to it are unreadable due to protections.

NAME

stty - Version 7 set terminal control information

DESCRIPTION

The 'stty' system call allows the user to change a terminal's current input/output configuration. This has been replaced by the ioctl system call, however interface routines exist for compatability with Version 6 and Version 7 Unix. It is highly recommended that programs use the new ioctl calls rather than the interface routines.

The tty interface buffer has the following structure:

```
struct tty {
    char t_ispeed;      input baud rate (see below)
    char t_ospeed;      output baud rate
    char t_erase;       erase character
    char t_kill;        kill character
    int t_modes;        bit masks for modes
};
```

The speed bytes change the speed of the terminal if the device is capable of changes. Changing the speed on a device not so equipped or to a speed that is not implemented for that device is useless. The following table shows the correspondence between the speed settings and the actual baud rates implemented:

B0	0	0 baud (hang up)
B50	1	50 baud
B75	2	75 baud
B100	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	19200 baud
EXTB	15	38400 baud

The terminal mode word sets the way REGULUS handles certain terminal dependent operations, including character mapping, echo, tabs, etc. The bit masks and their meanings are as follows:

ALLDELAY	0177400	All delay settings bits
BSDELAY	0100000	Backspace Delay
BS0	0000000	No backspace delay
BS1	0100000	Delay approximately .10 sec.
VTDELAY	0040000	Form feed delay

FF0	0000000	No form feed delay
FF1	0040000	Delay approximately .10 sec.
CRDELAY	0030000	Carriage return delay
CR0	0000000	No carriage return delay
CR1	0010000	Delay approximately .10 sec.
CR2	0020000	Delay approximately .20 sec.
CR3	0030000	Delay approximately .30 sec.
TBDELAY	0006000	Tab delay
TAB0	0000000	No tab delay
TAB1	0002000	Delay approximately .10 sec.
TAB2	0004000	Delay approximately .20 sec.
XTABS	0006000	Translate tabs to spaces
NLDELAY	0001400	New line (line feed) delay
NL1	0000400	Delay approximately .10 sec.
NL2	0001000	Delay approximately .20 sec.
NL3	0001400	Delay approximately .30 sec.
EVENP	0000200	Even parity
ODDP	0000100	Odd parity
RAW	0000040	Raw mode, wake up on all characters
CRMOD	0000020	map LF to CR-LF, echo CR or LF as CR-LF
ECHO	0000010	Echo input
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Same as raw, but handle INTR and QUIT
TANDEM	0000001	Enable XON-XOFF control

Delay bits specify one of a number of delays. A value of zero is used for no delay. All other values are system dependent.

Raw mode indicates that each character is to be passed to the reading process as soon as it is available. No special handling of the end-of-file (EOT), interrupt (DEL), or backspace is done. The line editor must be disabled with raw mode.

Cbreak mode is like raw mode on input in that the process is awakened on each character, but it also does kill and interrupt processing. No erase or input mapping is done. All output processing is still done. The cbreak mode is incompatible with raw mode and the line editor, so only one of the three should be used at any one time.

REGULUS will map carriage returns and line feeds into a carriage return /linefeed pair when enabled.

REGULUS will map uppercase characters into their lowercase equivalents if needed for older terminals.

The tab to blanks mapping bit is required to use other than hardware specified tab sizes.

CALLING SEQUENCE

```
#include <sgtty.h>
int stty(fd,ttybuf)     returns -1 for error, 0 otherwise
int fd;                 file descriptor of char special file
struct tty *ttybuf;     pointer to buffer to tty parameters
```

SEE ALSO

stty (cmnd)
ioctl (sys)
gtty (v7/sys)
ioctl (v7/sys)

RETURN VALUES

Zero is returned if there were no errors. -1 is returned if the file descriptor does not refer to an open terminal device or the user does not have write permission on that device.

NAME

times - Version 7 return execution times of task

DESCRIPTION

The 'times' system call is used to calculate the execution times of a task and its sub-tasks (children). All times are in 1/60's of a second, and are probably accurate to one second. The children's time is the sum of the task's currently terminated child tasks. This merely modifies the return values from the 'times (sys)' system call so that the time increments are 1/60's of a second rather than 1/10's of a second.

CALLING SEQUENCE

```
int times(tbuf)          returns -1 for error, 0 otherwise
long tbuf[4];            buffer in user space, contents are:
                             tbuf[0]: task user time
                             tbuf[1]: task system time
                             tbuf[2]: children user time
                             tbuf[3]: children system time
```

SEE ALSO

times (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the buffer address is invalid.

NAME
getlogin - get login name

SYNOPSIS
```
char *
getlogin()
```

DESCRIPTION
'getlogin' get's the current terminals login name as it is stored in the utmp file and returns a pointer to it.

FILES
/etc/utmp

DIAGNOSTICS
returns a -1 if the utmp file can not be opened.

SEE ALSO

NAME

acct - System III enable accounting record keeping

DESCRIPTION

The 'acct' system call, which outputs system accounting records on a per-task basis to an accounting file, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

chown - System III change owner of a file

DESCRIPTION

The 'chown' system call is used to change the owner of a file. The effective user id of the process must be the owner of the file or the super-user. The numerical ownerid and groupid of the named file is changed to the specified ids. Care should be taken that the new id has been assigned to a user. This is implemented as calls to 'chown (sys)' and 'chgrp (sys)'.

CALLING SEQUENCE

```
int chown(fname,uid,gid)    returns -1 for error, 0 otherwise
char *fname;                name of file to change owner
int uid;                    new owner id of file
int gid;                    new group id of file
```

SEE ALSO

chown (sys)
chgrp (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if you are not the super-user.

NAME

fstat - System III get file status by file descriptor

DESCRIPTION

The 'stat' system call is used to get file information about the file whose open descriptor is specified. This is implemented as a function that does a REGULUS 'fstat (sys)' system call, then rearranges the returned values.

The inode number is the REGULUS index record number truncated to 16 bits and the inode changed time is the REGULUS modified date.

CALLING SEQUENCE

```
#include <stat.h>
int fstat(fd,buf)       returns -1 for error, 0 otherwise
int fd;                 file name of file to stat
struct stat *buf;       pointer to stat buffer

/* stat structure for Unix System III */

struct stat {
    int st_dev;         /*device inode is on*/
    int st_ino;         /*inode number*/
    int st_mode;        /*file mode bits*/
    int st_nlink;       /*number of links*/
    int st_uid;         /*owner id*/
    int st_gid;         /*group id*/
    int st_rdev;        /*real device if mount, char or block*/
    long st_size;       /*file size in bytes*/
    long st_atime;      /*access time*/
    long st_mtime;      /*modified time*/
    long st_ctime;      /*inode changed time*/
};

#define    S_IFMT     0170000    /*file type*/
#define    S_IFIFO    0010000    /*fifo special*/
#define    S_IFCHR    0020000    /*character*/
#define    S_IFDIR    0040000    /*directory*/
#define    S_IFBLK    0060000    /*block*/
#define    S_IFREG    0100000    /*regular*/
#define    S_ISUID    0004000    /*set user id*/
#define    S_IGUID    0002000    /*set group id*/
#define    S_ISVTX    0001000    /*save text*/
#define    S_IREAD    0000400    /*owner read*/
#define    S_IWRITE   0000200    /*owner write*/
#define    S_IEXEC    0000100    /*owner exec*/
```

SEE ALSO

stat (sys3/sys)
fstat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the named file does not exist or if any of the directories

leading to it are unreadable due to protections.

NAME

mknod - System III make a special file or directory

DESCRIPTION

The 'mknod' system call is used to make special file entries in the filesystem. Special file entries are those for devices and directories. The mode bits passed to the system call determine whether you are making a special file corresponding to a device or directory. The entry is made with protections that are the specified protections 'anded' with the complement of the current user's umask. This interface routine alters the System III special file mode bits to match the REGULUS special file mode bits.

The access mode bits are discussed in the chmod (cmnd) document. The special device mode bits are defined in 'stat.h'.

If the mode bits specify a directory, block, or character special file only the super-user may execute this call. If a named pipe (or fifo) file is specified, any user may execute this call.

CALLING SEQUENCE

```
#include <stat.h>

int mknod(fname,mode,info)     returns -1 if error, 0 otherwise
char *fname;                   name of new special file or directory
int mode;                      mode bits for special file
int info;                      primary and secondary device numbers for
                               device special files (ignored for directories)
```

SEE ALSO

mkdir (cmnd)
mknod (cmnd)
umask (sys)
filesys (files)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the node could not be created.

NAME

nice - System III change task priority

DESCRIPTION

The nice System III interface routine attempts to make a reasonable mapping of System III nice values to REGULUS nice values. On REGULUS, priorities are always positive, with larger numbers implying higher priority. On System III, negative priorities are for system tasks and the smaller the priority value, the higher the priority.

CALLING SEQUENCE

```
int nice(incr)      returns -1 for error, new priority otherwise
int incr;           increment to add to current priority
```

SEE ALSO

nice (cmnd)
ps (cmnd)
nice (sys)

RETURN VALUES

The new actual task priority is returned if there was no error. -1 is returned if the user is not allowed to change the priority to the resulting value.

NAME

profil - System III profile task execution

DESCRIPTION

The 'profil' system call, which is used to get an execution profile of a task, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

stat - System III get file status by file name

DESCRIPTION

The 'stat' system call is used to get file information about the file whose path name is specified. This is implemented as a function that does a REGULUS 'stat (sys)' system call, then rearranges the returned values.

The inode number is the REGULUS index record number truncated to 16 bits and the inode changed time is the REGULUS modified date.

CALLING SEQUENCE

```
#include <stat.h>
int stat(fn,buf)        returns -1 for error, 0 otherwise
char *fn;               file name of file to stat
struct stat *buf;       pointer to stat buffer

/* stat structure for Unix System III */

struct stat {
    int st_dev;         /*device inode is on*/
    int st_ino;         /*inode number*/
    int st_mode;        /*file mode bits*/
    int st_nlink;       /*number of links*/
    int st_uid;         /*owner id*/
    int st_gid;         /*group id*/
    int st_rdev;        /*real device if mount, char or block*/
    long st_size;       /*file size in bytes*/
    long st_atime;      /*access time*/
    long st_mtime;      /*modified time*/
    long st_ctime;      /*inode changed time*/
};

#define     S_IFMT      0170000     /*file type*/
#define     S_IFIFO     0010000     /*fifo special*/
#define     S_IFCHR     0020000     /*character*/
#define     S_IFDIR     0040000     /*directory*/
#define     S_IFBLK     0060000     /*block*/
#define     S_IFREG     0100000     /*regular*/
#define     S_ISUID     0004000     /*set user id*/
#define     S_IGUID     0002000     /*set group id*/
#define     S_ISVTX     0001000     /*save text*/
#define     S_IREAD     0000400     /*owner read*/
#define     S_IWRITE    0000200     /*owner write*/
#define     S_IEXEC     0000100     /*owner exec*/
```

SEE ALSO

fstat (sys3/sys)
stat (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the named file does not exist or if any of the directories

leading to it are unreadable due to protections.

NAME

times - System III return execution times of task

DESCRIPTION

The 'times' system call is used to calculate the execution times of a task and its sub-tasks (children). All times are in 1/60's of a second, and are probably accurate to one second. The children's time is the sum of the task's currently terminated child tasks. This merely modifies the return values from the 'times (sys)' system call so that the time increments are 1/60's of a second rather than 1/10's of a second.

CALLING SEQUENCE

```
int times(tbuf)          returns -1 for error, 0 otherwise
long tbuf[4];            buffer in user space, contents are:
                             tbuf[0]: task user time
                             tbuf[1]: task system time
                             tbuf[2]: children user time
                             tbuf[3]: children system time
```

SEE ALSO

times (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the buffer address is invalid.

NAME

ulimit - System III establish user resource limits

DESCRIPTION

The 'ulimit' system call, which limits some user resources, such as maximum file size, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

NAME

uname - System III get system names

DESCRIPTION

The 'uname' system call obtains the system name, version string, release string and node name from the currently executing kernel. This is implemented as REGULUS 'sysinfo (sys)' system calls.

CALLING SEQUENCE

```
#include <sys/utsname.h>
int uname(ubuf)          returns -1 for error, 0 otherwise
struct utsname *name;

/* The utsname structure returned from uname */

struct utsname {
    char sysname[9];
    char nodename[9];
    char release[9];
    char version[9];
};
```

SEE ALSO

uname (cmnd)
sysinfo (sys)

RETURN VALUES

Zero is returned if there was no error. -1 is returned if the buffer address is invalid.

NAME

ustat - System III get status of mounted device

DESCRIPTION

The 'ustat' system call, which obtains the status of a mounted device, is unimplemented in REGULUS.

RETURN VALUES

-1 is returned always

SUBROUTINES

This section describes most of the subroutines available on REGULUS. Their binary versions reside in system libraries in the directory called /lib. File lib7.a contains almost all the subroutines, including those for standard I/O (UNIX Version 7). (Standard I/O subroutines are described in Standard I/O section of this manual.)

Subroutines are described in this section in the following sequence:

NAME	Name of the subroutine
SYNOPSIS	Syntax for calling the subroutine.
DESCRIPTION	Briefly describes the results or purpose of the subroutine.
FILES	Lists the file names that are built into the program.
DIAGNOSTICS	Describes routine for diagnosing programming errors.
SEE ALSO	Gives pointers to related information.

NAME

alloc - C storage allocator

SYNOPSIS

```
char *alloc(nbytes)
unsigned nbytes;
```

DESCRIPTION

'alloc' allocates memory from the 'break' area of the task. 'alloc' uses a circular first fit algorithm for memory allocation and returns a pointer to a memory area which is at least nbytes in size. If there is not enough memory available, a 0 is returned.

FILES

DIAGNOSTICS

If a memory area of the correct size can not be allocated a 0 will be returned.

SEE ALSO

calloc(subs), free(subs)

NAME

amatch - regular expression pattern matcher

SYNOPSIS

```
char *amatch(line,ptr,pat)
char *line, *ptr, *pat;
```

DESCRIPTION

'amatch' is a general purpose regular expression pattern match routine. 'amatch' returns a pointer to the position in the string 'line' where a match of 'pat' ends, starting from 'ptr', or zero if the pattern does not match starting from 'ptr'. For example:

```
line = "this is a test"
ptr = &line[2]
pat =  "?*[a-z] s"
```

amatch would return:

```
&line[12]
```

FILES

/usr/include/pat.h regular expression special character defines

DIAGNOSTICS

SEE ALSO

match (subs)
onematch (sub)
ed (cmnd) for a description of regular expressions

NAME

atof - ascii to floating point conversion

SYNOPSIS

```
#include <math.h>

double
atof(s)
char *s;
```

DESCRIPTION

'atof' converts an ascii string into it's floating point representation, where the string is of the format :

{sign}{digits}{'.'}{digits}{E}{sign}{digits}

Both signs and the exponent string is optional. The decimal point is optional, but may appear at any point in the digit string. The single precision result is returned.

FILES

math.h - routine declaration

SEE ALSO

etoa(subs), ftoa(subs), atol(subs)

DIAGNOSTICS

NAME
atoi - ascii to integer conversion

SYNOPSIS
```
int atoi(s);
char *s;
```

DESCRIPTION
'atoi' converts a null terminated ascii string into its numeric equivalent. The number may be signed. The conversion will stop at the first non-decimal digit although they may be preceded by spaces, tabs and carriage returns.

FILES

SEE ALSO
itoa(subs), atol(subs), ltoa(subs)

NAME

atol - ascii to long numeric conversion

SYNOPSIS

```
long atol(s);
char *s;
```

DESCRIPTION

'atol' converts an ascii string into a long number. The number may optionally be signed. Conversion will stop at the first non-decimal digit or at the null terminator. The number may be preceded by spaces, tabs and carriage returns.

FILES

SEE ALSO

itoa(subs), atoi(subs), ltoa(subs)

NAME

atoo - ascii to octal conversion

SYNOPSIS

```
int atoo(s);
char *s;
```

DESCRIPTION

'atoo' converts a null terminated ascii string into its octal equivalent. The number may be signed. Does not require a preceding zero. The conversion will stop at the first non-octal digit although they may be preceded by spaces, tabs and carriage returns.

FILES

SEE ALSO

atoi(subs), itoa(subs), atol(subs), ltoa(subs)

NAME
calloc - C memory allocator

SYNOPSIS
```
char *calloc(n,s);
int n,s;
```

DESCRIPTION
'calloc' allocates n blocks of memory, where s is the size of the desired blocks.

DIAGNOSTICS
If a memory area of the correct size can not be allocated a 0 will be returned.

SEE ALSO
alloc (subs), free (subs)

NAME
ceil - floating point ceiling function

SYNOPSIS
```
#include <math.h>

double ceil(d)
double d;
```

DESCRIPTION
'ceil' returns the smallest integer (as a double precision number) not less than x.

FILES
math.h - declaration include file

SEE ALSO
floor(subs)

NAME

ckfpat - check full path

SYNOPSIS

```
ckfpat (fpat,dirname);
char *fpat,*dirname;
```

DESCRIPTION

'ckfpat' accepts a pathname in 'fpat', strips the directory path off and returns it in 'dirname'. If 'fpat' does not contain a directory path 'ckfpat' assumes the directory to be the current working directory and returns the directory as ".".

FILES

/usr/include/pat.h the regular expression special character defin

DIAGNOSTICS

SEE ALSO

fmatch (subs), dofmatch (subs)

NAME

cryptpw - encrypt a password

SYNOPSIS

```
char *cryptpw(str)
char *str;
```

DESCRIPTION

'cryptpw' encrypts the string 'str' and returns a pointer to the resulting eight character string. The password is padded with random characters to get eight characters, these characters are permuted using a permutation table and translated using a translation table.

FILES

SEE ALSO

pwcmp (subs)

NAME
ctime - convert time and date to ascii

SYNOPSIS
```
char *ctime(tvec);
long *tvec;
```

DESCRIPTION
'ctime' converts a time as returned by 'time (sys)' into a string. 'ctime' returns a pointer to a 26 byte static array which contains the time and date. The array is in the form: 'dwk mon dy hr:mi:se year\\n'. This produces a date which looks like: Mon Jan 1 12:30:45 1970\\n.

FILES

DIAGNOSTICS

SEE ALSO
time (sys), gmtime (subs), localtime (subs)

NAME

doccl - do character class pattern matching

SYNOPSIS

```
doccl(pat, chr)
char *pat;
char chr;
```

DESCRIPTION

'doccl' returns a 1 if the given character occurs in the character class expression pat. The character class expression must begin with the character '[' and end with the character ']'. A zero is returned if the character is not in the character class.

FILES

/usr/include/pat.h regular expression special character defines

DIAGNOSTICS

SEE ALSO

amatch (sub)
match (sub)
onematch (sub)
ed (cmnd) for a description of regular expressions

NAME

dofmatch - file pattern matching routine

SYNOPSIS

```
dofmatch(dirname,pat);
char *dirname,*pat;
```

DESCRIPTION

'dofmatch' tries to find filenames in the specified directory which match the given pattern. 'dofmatch' returns the number of matches which occurred.

FILES

/usr/include/pat.h the regular expression special character defin

DIAGNOSTICS

SEE ALSO

fmatch (subs), ckfpat (subs), match (subs)

NAME

etoa - floating point exponential to ascii

SYNOPSIS

```
#include <math.h>

char *
etoa(fp,buf,prec)
float fp;
char *buf;
int prec;
```

DESCRIPTION

'etoa' converts a float into it's ascii exponential representation. Where fp is a floating point number, buf is the buffer in which to return the string, and prec is the precision of the decimal places. If the precision is specified to be zero or negative then the default precision of six decimal places will be used. A pointer to the beginning of buf is returned.

FILES

math.h - routine declaration

SEE ALSO

ftoa(subs), atoi(subs), atol(subs), itoa(subs)

DIAGNOSTICS

NAME

exp - computes the exponential function

SYNOPSIS

```
#include <math.h>

double
exp(d)
double d;
```

DESCRIPTION

'exp' returns the results of the exponential function e ^ x (where e is 2.718...).

FILES

math.h - declarations of math routines

SEE ALSO

NAME
fabs - floating point absolute

SYNOPSIS
```
#include <math.h>

double
fabs(d)
double d;
```

DESCRIPTION
'fabs' returns the absolute value of a floating point number.

FILES
math.h - declaration include file

SEE ALSO

NAME

findexec - find executable program

SYNOPSIS

```
findexec(pname,path,fname)
char *pname, *path, *fname;
```

DESCRIPTION

'findexec' attempts to find the executable file of the name 'pname'. It searches first in the current directory, and then on the directories specified in path. The directories specified in path should be separated by colons (:). If it is successful then a 1 is returned and the pathname is returned in fname, otherwise a zero is returned.

SEE ALSO

NAME
floor - floating point floor function

SYNOPSIS
```
#include <math.h>

double floor(d)
double d;
```

DESCRIPTION
'floor' returns the largest integer (as a double precision number) not greater than x.

FILES
math.h - declaration include file

SEE ALSO
ceil(subs)

NAME

fmatch - pattern matching routine

SYNOPSIS

```
fmatch(str,pat)
char *str;
char *pat;
```

DESCRIPTION

'fmatch' compares the string 'str' to the pattern 'pat' and returns a 1 if a match occurs. The pattern may include asterisks '*', question marks '?' and bracket pairs '[' ']'. An asterisk will match any number of characters. A question mark will match any single character, and the characters enclosed in brackets are a set of legitimate characters for matching. A zero is returned if the line does not match the pattern.

FILES

/usr/include/pat.h the regular expression special character defin

DIAGNOSTICS

SEE ALSO

dofmatch (subs), ckfpat (subs), match (subs)

NAME

fmod - floating point remainder function

SYNOPSIS

```
#include <math.h>

double fmod(x,y)
double x,y;
```

DESCRIPTION

'fmod' returns the remainder when x is divided by y.

FILES

math.h - declaration include file

SEE ALSO

NAME

free - free memory

SYNOPSIS

```
free(ap);
char *ap;
```

DESCRIPTION

'free' returns the block of memory associated with ap to the freelist. This memory should previously have been allocated by 'alloc (subs)' or 'calloc (subs)'.

SEE ALSO

calloc (subs), alloc (subs)

NAME

ftoa - floating point to ascii conversion

SYNOPSIS

```
#include <math.h>

char *
ftoa(fp,buf,prec)
float fp;
char *buf;
int prec;
```

DESCRIPTION

'ftoa' converts a float into it's ascii representation. Where fp is a floating point number, buf is the buffer in which to return the string, and prec is the precision of the decimal places. If the specified precision is zero then no decimal point will be printed. If the precision is negative then the default precision (6) will be used. A pointer to the beginning of buf is returned.

FILES

math.h - routine declaration

SEE ALSO

etoa(subs), atoi(subs), atol(subs), itoa(subs)

DIAGNOSTICS

NAME

getarhd - get archive header

SYNOPSIS

```
#include <ar68.h>
getarhd(fp,arptr)
FILE *fp;
struct libhdr *arptr;
```

DESCRIPTION

'getarhd' fills the archive header structure, pointed to by 'arptr', from the input/output device fp. Input/output independent mapping routine which builds the structure in a way which will be understood by the current machine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

putarhd (subs)

NAME

getchd - get a.out header

SYNOPSIS

```
#include <cout.h>
getchd(fp,ptr)
FILE *fp;
struct hdr *ptr;
```

DESCRIPTION

'getchd' fills the cout header structure, pointed to by 'ptr', from the input/output device fp. Input/output independent mapping routine which builds the structure in a way which will be understood by the current machine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

putchd (subs)

NAME
getegid - get effective group id

SYNOPSIS
```
getegid();
```

DESCRIPTION
'getegid' returns the effective group id of the calling process.

SEE ALSO
getgid (sys)

NAME

getenv - get value for environment variable

SYNOPSIS

```
char *getenv(name);
char *name;
```

DESCRIPTION

'getenv' searches the environment for the specified name. If the name is found, a pointer to its corresponding value is returned. If the name is not found, NULL is returned. This relies upon the environment information passed to the task on the stack not being altered.

SEE ALSO

env (cmnd)
environment (misc)

NAME
geteuid - get effective user id

SYNOPSIS
```
geteuid();
```

DESCRIPTION
'geteuid' returns the effective user id of the calling process.

SEE ALSO
getuid (sys)

NAME
getppid - get parent's process id.

SYNOPSIS
getppid()

DESCRIPTION
'getppid' returns the parent process's process id, this is the high byte of the high word returned by getpid.

SEE ALSO
getpid(sys)

NAME

getpw - get password entry

SYNOPSIS

```
getpw(uid,buf)
int uid;
char *buf;
```

DESCRIPTION

'getpw' returns the entry in the password file associated with the specified user id (uid). 'buf' must be large enough to hold eighty characters. If an entry is found for the specified user id it is copied to buf and 'getpw' returns 0, otherwise 'getpw' returns 1. The password file is not closed.

FILES

/etc/passwd (password file)

SEE ALSO

passwd (files)

NAME
gettime - get time of day

SYNOPSIS
```
long gettime();
```

DESCRIPTION
'gettime' returns the time of day.

SEE ALSO
time (sys)

NAME
index - string position function

SYNOPSIS
```
index(s,ch);
char *s,ch;
```

DESCRIPTION
'index' searches for the character ch in the string s. If the character exists, the position in the string is returned. Otherwise a -1 is returned.

FILES

SEE ALSO
rindex(subs)

NAME

itoa - integer to ascii conversion

SYNOPSIS

```
int itoa(n,str,width)
int n, width;
char *str;
```

DESCRIPTION

'itoa' converts an integer into its ascii equivalent. The string is blank padded on the left to make it 'width' characters. The string is returned in the second argument.

FILES

SEE ALSO

atoi(subs), atol(subs), ltoa(subs)

BUGS

The number is limited to 5 characters (a true 16 bit integer).

NAME
ldiv - long division

SYNOPSIS
```
long ldiv(l1,l2);
long l1,l2;
```

DESCRIPTION
'ldiv' computes the division of 'l1' by 'l2' and returns the quotient.

SEE ALSO
lmul (subs), lrem (subs)

NAME

ledit - write line to line editor buffer

SYNOPSIS

```
ledit(fd,str)
int fd;
char *str;
```

DESCRIPTION

'ledit' writes the given string into the line editor buffer of the open file associated with the given file descriptor. The string is then available to the user for modification with the line editor commands.

The file descriptor should point to a terminal with ICANNON set. 'ledit' places a newline character at the end of the string before writing.

FILES

DIAGNOSTICS

'ledit' returns 1 if successful, 0 if unsuccessful.

SEE ALSO

NAME

lgetl, lgetw - get a out word or long

SYNOPSIS

```
lgetl(l,fp)
long *l;
FILE *fp;

lgetw(w,fp)
short *w;
FILE *fp;
```

DESCRIPTION

'lgetl' gets a long from the specified input/output device in the byte order required by the target machine. 'lgetw' gets a a 16-bit word from the input/output FILE. Input/output independent mapping routine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

lputl (subs)

NAME
lmul - long multiplication

SYNOPSIS
```
long lmul(l1,l2);
long l1,l2;
```

DESCRIPTION
'lmul' computes the long multiplication on 'l1' and 'l2' and returns the product.

SEE ALSO
ldiv (subs), lrem (subs)

NAME
log - compute the logarithm of a number

SYNOPSIS
```
#include <math.h>

double
log(d)
double d;
```

DESCRIPTION
'log' returns the computed logarithm of it's argument.

FILES
math.h - declarations of math routines

SEE ALSO

NAME

lputl, lputw - put out word or long

SYNOPSIS

```
lputl(l,fp)
long *l;
FILE *fp;

lputw(w,fp)
short *w;
FILE *fp;
```

DESCRIPTION

'lputl' puts a long out to the input/output device using putc in the byte order required by the target machine (68000). 'lputw' writes out a 16-bit word to the input/output FILE. Input/output independent mapping routine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

lgetl (subs)

NAME

lreadl, lreadw - read in a word or long

SYNOPSIS

```
lreadl(fd,l)
int fd;
long *l;

lreadw(fd,w)
int fd;
short *w;
```

DESCRIPTION

'lreadl' reads a long in from the specified file descriptor in the byte order required by the target machine. 'lreadw' reads a a 16-bit word from the specified file descriptor. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

lwritel (subs)

NAME
lrem - long remainder

SYNOPSIS
```
long lrem(l1,l2);
long l1,l2;
```

DESCRIPTION
'lrem' does long division on the 'l1', 'l2' pair and returns the remainder from 'l1' divided by 'l2'.

FILES

SEE ALSO
lmul (subs), ldiv (subs)

NAME

lsbrk - add to task break

SYNOPSIS

```
char *lsbrk(incr);
int incr;
```

DESCRIPTION

'lsbrk' allows adding incremental amounts of memory to a task. 'lsbrk' works the same as 'brk (sys)', except that the value specified is added to the current break pointer. 'lsbrk' returns a pointer to the new memory.

SEE ALSO

brk (sys)

NAME

ltoa - long to ascii conversion

SYNOPSIS

```
ltoa(n,str,width)
long n;
char *str;
int width;
```

DESCRIPTION

'ltoa' converts a long into its ascii equivalent. The string is blank padded on the left to make it 'width' characters. The string is returned in the second argument.

FILES

SEE ALSO

atoi(subs), atol(subs), itoa(subs)

BUGS

The number is limited to 11 characters.

NAME

lwritel, lwritew - write out a word or long

SYNOPSIS

```
lwritel(fd,l)
int fd;
long *l;

lwritew(fd,w)
int fd;
short *w;
```

DESCRIPTION

'lwritel' writes a long to the specified file descriptor in the byte order required by the target machine. 'lwritew' writes a a 16-bit word to the specified file descriptor. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

lreadl (subs)

NAME
match - match a regular expression pattern

SYNOPSIS
```
match(line,pat)
char *line;
char *pat;
```

DESCRIPTION
'match' returns a 1 if the line contains a match for the regular expression pattern pat. A zero is returned if the line does not match the pattern.

FILES
/usr/include/pat.h the regular expression special character defines

DIAGNOSTICS

SEE ALSO
amatch (subs)
ed (cmnd) for a description of regular expressions

NAME

math - declaration of the math routines

SYNOPSIS

#include <math.h>

DESCRIPTION

'math.h' contains the declarations necessary for the floating point mathematical routines, as well as the floating point and long conversion routines.

SEE ALSO

atol(subs), ltoa(subs), atof(subs), ftoa(subs), etoa(subs), sin(s
fabs(subs), log(subs), pow(subs), sqrt(subs), exp(subs)

NAME

mktemp - make a temporary file name

SYNOPSIS

```
mktemp(file)
char *file;
```

DESCRIPTION

'mktemp' changes the last five characters of 'file' in order to create a unique file name. The characters will be replaced with the current pid (in octal) and a letter. The letter allows the same process to generate a number of different temporary file names.

NAME

nextpat - advance to next pattern

SYNOPSIS

```
char *nextpat(str)
char *str;
```

DESCRIPTION

'nextpat' returns a pointer to the next regular expression pattern in the string or zero if there are no more patterns.

FILES

/usr/include/pat.h regular expression special character defines

DIAGNOSTICS

SEE ALSO

amatch (sub)
match (sub)
onematch (sub)
ed (cmnd) for a description of regular expressions

NAME
onematch - general pattern match

SYNOPSIS
```
onematch(line,ptr,pat)
char *line, *ptr, *pat;
```

DESCRIPTION
'onematch' is a general purpose regular expression pattern match routine. It returns the count from 'ptr' in 'line' of a match of the regular expression 'pat'. Zero is returned if there is no match.

FILES
/usr/include/pat.h regular expression special character defines

DIAGNOSTICS

SEE ALSO
amatch (sub)
match (sub)
ed (cmnd) for a description of regular expressions

NAME

pow - floating point power function

SYNOPSIS

```
#include <math.h>

double
pow(x,y)
double x, y;
```

DESCRIPTION

'pow' computes the value of 'x' raised to the power of 'y', and returns the results.

FILES

math.h - declarations of math routines

SEE ALSO

NAME

printf - formatted print

SYNOPSIS

```
printf(plist);
char *plist;
```

DESCRIPTION

'printf' takes the input string and interprets various formatting commands and outputs the results to the standard output device. There are two types of items in the input string: characters which are copied literally and format statements which work on strings, characters and numerics.

The string takes the form of a literal string with embedded format statements and the various arguments on which the format statements act: printf("...",arg1,arg2...). All format statements are preceded by a percent sign and terminated by one of the conversion characters. Between the percent sign and the conversion characters are optional symbols which adjust the standard formats.

The format statement consists of a contiguous group of characters. A minus sign '-' may follow the percent sign to designate that the item is to be left justified. A decimal number designating the field width may optionally be specified. If the item is larger than the specified width it will be printed as is. If the item is smaller, the item will be padded with spaces. If the first digit of the specified width is a '0' then the item will be padded with zero's instead of spaces. If the item is a string or a floating point number a second field preceded by a period '.' may be specified. The second width field specifies the number of digits to the right of the decimal point in the case of a floating point and the maximum number of characters to be printed in the case of a string.

The last item of a format statement describes the item we are formatting. They are 'c', 'd', 'e', 'f', 'u', 'o', 's', and 'x'.

'c' The argument to be printed is a single character.

's' The argument is a string or character pointer. All characters will be printed unless a maximum width field is in force.

'd' The argument is a decimal numeric.

'o' The argument is a octal numeric.

'x' The argument is a hexidecimal numeric.

'u' The argument will be interpreted as an unsigned decimal integer in the range 0 to 65535.

'f' The argument is either a floating point or double precision number. It will be of the form: an optional minus sign, digits, a decimal point, and more digits. If no second width field has been specified there will be six digits to the right of the decimal point.

'e' The argument is to be printed in scientific notation and is either a floating point or double precision number. The format is exactly like the 'f' format except that only one digit is to the right of the decimal point and a two digit exponent is specified after the number.

Format descriptors 'd', 'o', 'x', and 'u' may optionally be preceeded by a 'l' character to specify that the argument to be printed is a long rather than an integer.

Any other character following a percent sign will be taken as a literal and will itself be printed. In this way you can print a percent sign or a double quote.

SEE ALSO
scanf (subs)

NAME

putarhd - write archive header onto a input/output device

SYNOPSIS

```
#include <cout.h>
putarhd(fp,arptr)
FILE *fp;
struct libhdr *arptr;
```

DESCRIPTION

'putarhd' writes the library header structure, pointed to by 'arptr', onto the input/output device fp. Input/output independent mapping routine which outputs the structure in a fashio which will be understood by the target machine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

getarhd (subs)

NAME

putchd - write a.out header onto a input/output device

SYNOPSIS

```
#include <cout.h>
putchd(fp,ptr)
FILE *fp;
struct hdr *ptr;
```

DESCRIPTION

'putchd' writes the cout header structure, pointed to by 'ptr', onto the input/output device fp. Input/output independent mapping routine which outputs the structure in a fashio which will be understood by the target machine. It returns 0 for success, -1 for failure.

DIAGNOSTICS

Returns 0 for success, -1 for failure.

SEE ALSO

getchd (subs)

NAME

pwcmp - compare passwords

SYNOPSIS

```
pwcmp(passwd,epasswd)
char *passwd,*epasswd;
```

DESCRIPTION

'pwcmp' encrypts the first password (passwd) and compares the result with the second password (epasswd), which has already been encrypted. If the compare fails 'pwcmp' returns 1, if it succeeds 'pwcmp' returns 0. The return parameters are compatible with 'strcmp (subs)'.

FILES

SEE ALSO

crypt (subs)

NAME

qsort - general sort routine

SYNOPSIS

```
qsort(base, n, width, compar);
char *base;
int n, width;
int (*compar)();
```

DESCRIPTION

'qsort' sorts the items according to a designated order. 'base' is a character string of items. 'n' is the number of items. 'width' is how wide each item is. 'compar' is your own routine which decides whether two items are equivalent, greater than or less than one another.

SEE ALSO

strcmp (subs)

NAME

rand - random number generator
srand - set random number generator

SYNOPSIS

```
int rand()

srand(seed)
int seed;
```

DESCRIPTION

'rand' is a routine which returns successive pseudo-random numbers in the range from 0 to 32767. This sequence is reinitialized by calling 'srand' with an argument ('seed') of 1. It can be set to a random starting point by calling 'srand' with the low-order word of the time.

NAME

readrclock - read read-time clock

SYNOPSIS

```
#include <time.h>
struct tm *clock();
```

DESCRIPTION

'readrclock' reads the real-time clock and returns the information as a standard gmtime() structure. This routine will transfer all information that is available from the real-time clock into the "tm" structure. Any information that is not available DIRECTLY by reading the real-time clock will be set to 0 (e.g. the year and number of days since the beginning of the year is not available on some real-time clocks).

FILES

/dev/mem - used to read the real-time clock.

DIAGNOSTICS

If for any reason, readrclock cannot read the real-time clock, it will return a 0.

SEE ALSO

setrclock(subs)

NAME

rindex - string position function

SYNOPSIS

```
rindex(s,ch);
char *s,ch;
```

DESCRIPTION

'rindex' searches for the character ch in the string s. If the character exists, the position of the last occurance of the character in the string is returned. Otherwise a -1 is returned.

FILES

SEE ALSO

index(subs)

NAME

setjmp - set user stack for non-local goto
longjmp - non-local goto

SYNOPSIS

```
int setjmp(env)
long env[3];

int longjmp(env, return)
long env[3];
int return;
```

DESCRIPTION

'setjmp' saves its local environment in the array 'env' for use later by a call to 'longjmp'. By passing this environment in a call to 'longjmp', you effect a non-local goto.

The call to 'longjmp' simulates a return from the last call to 'setjmp'. The environment from the call to 'setjmp' must still exist. This means that you may not have returned from the procedure in which 'setjmp' is called. These routines are useful in error recovery. The return value from 'setjmp' when it is called is always zero.

The parameter 'return' is what will be used as the return value from 'setjmp' when 'longjmp' is called. If that value is zero, the system will coerce a 1 return value automatically.

FILES

DIAGNOSTICS

SEE ALSO

NAME
setrclock - set the real-time clock

SYNOPSIS
```
#include <time.h>
setrclock(clock)
struct tm clock;
```

DESCRIPTION
'setrclock' sets the real-time clock to the date and time specified in the "tm" time structure. A 1 is returned when setrclock is successful.

FILES
/dev/mem - used to set the real-time clock.

DIAGNOSTICS
If for any reason, setrclock cannot set the real-time clock, it will return a 0.

SEE ALSO
readrclock(subs)
time(sys)

NAME

sin - sine function

SYNOPSIS

```
#include <math.h>

double
sin(d)
double d;

double
cos(d)
double d;

double
tan(d)
double d;
```

DESCRIPTION

These compute the sine, cosine, and tangent of a given floating point angle. The argument is the number of radians.

FILES

math.h - declaration include file.

SEE ALSO

NAME

sleep - sleep for specified time

SYNOPSIS

```
int sleep(seconds)      returns -1 for error, 0 otherwise
int seconds;            number of seconds to sleep
```

DESCRIPTION

The 'sleep' subroutine is a replacement of the sleep system call. 'sleep' uses the alarm and pause system calls to put the process to sleep for the requested number of seconds. The actual amount of time may be longer than that requested due to scheduling activity. If the seconds requested was a second, the time may be less than a second due to the fact that sleeps are done in integrals of the real time clock. A previously set alarm signal is reset at the end of the sleep time. If an alarm was already set, and it is less that the sleep time, the sleep will return at the alarm time after a user specified signal catching routine is executed. If the previous alarm time is greater than the sleep time, then the alarm is reset to go off at the requested time.

SEE ALSO

signal (sys)
ssignal (sys)

BUGS

NAME
sqrt - square root function

SYNOPSIS
```
#include <math.h>

double
sqrt(d)
double d;
```

DESCRIPTION
'sqrt' returns the computed square root of it's argument.

FILES
math.h - declarations of math routines

SEE ALSO

NAME

strcat - string concatenation

SYNOPSIS

```
char *strcat(s,t);
char *s,*t;
```

DESCRIPTION

'strcat' concatenates the string 't' onto the end of string 's' after removing the null terminator. 's' must have been allocated enough memory or a string overflow may occur. returns pointer to concatenated string.

FILES

SEE ALSO

strcmp (subs), strcpy (subs), strlen (subs)

NAME
strchr - find character in string

SYNOPSIS
```
char *strchr(s,c)
char *s;
char c;
```

DESCRIPTION
'strchr' finds the first occurance of 'c' in the string 's' and returns a pointer to it. If the character is not found a NULL is returned.

SEE ALSO
strcmp (subs), strcpy (subs), strcat (subs)

NAME

strcmp - string compare

SYNOPSIS

```
strcmp(s1,s2);
char *s1, *s2;
```

DESCRIPTION

'strcmp' compares two strings and returns a 0 if the two are exactly the same, a positive number if 's1' is greater than 's2' and a negative number if 's1' is less than 's2'.

FILES

SEE ALSO

strcpy (subs), strlen (subs), strcat (subs)

NAME
strcpy - string copy

SYNOPSIS
```
char *strcpy(s1,s2)
char *s1,s2;
```

DESCRIPTION
'strcpy' copies 'S2' into 'S1' and returns 'S1'. 'S1' must already have memory allocated to it.

FILES

SEE ALSO
strcmp (subs), strlen (subs), strcat (subs)

NAME

strend - string end match

SYNOPSIS

```
strend(s1,s2);
char *s1, *s2;
```

DESCRIPTION

'strend' returns 1 if the end of 's1' matches 's2', 0 otherwise.

FILES

SEE ALSO

strcpy (subs), strlen (subs), strcat (subs), strcmp (subs)

NAME
strlen - string length

SYNOPSIS
```
strlen(s);
char *s;
```

DESCRIPTION
'strlen' computes and returns the length of the string 's' up to but not including the null terminator.

FILES

SEE ALSO
strcmp (subs), strcpy (subs), strcat (subs)

NAME

strncat - string concatenation

SYNOPSIS

```
char *strncat(s,t,n)
char *s,*t;
int n;
```

DESCRIPTION

'strncat' concatenates the string 't' onto the end of string 's' after removing the null terminator. The resulting string is at most 'n' characters long. 's' must have been allocated enough memory or a string overflow may occur. Returns pointer to concatenated string.

FILES

SEE ALSO

strcmp (subs), strcpy (subs), strlen (subs)

NAME

strncmp - string compare

SYNOPSIS

```
strncmp(s1,s2,n)
char *s1, *s2;
int n;
```

DESCRIPTION

'strncmp' compares two strings and returns a 0 if the two are exactly the same, a positive number if 's1' is greater than 's2' and a negative number if 's1' is less than 's2'. At most n characters are compared.

FILES

SEE ALSO

strcpy (subs), strlen (subs), strcat (subs)

NAME

strncpy - string copy

SYNOPSIS

```
char *strncpy(s1,s2,n)
char *s1,s2;
int n;
```

DESCRIPTION

'strncpy' copies 'S2' into 'S1' and returns 'S1'. 'S1' must already have memory allocated to it. At most 'n' characters are copied, and if 's2' is less than 'n' characters, 's1' is padded with nulls.

FILES

SEE ALSO

strcmp (subs), strlen (subs), strcat (subs)

NAME

strpbrk - find character set in string

SYNOPSIS

```
char *strpbrk(s1,s2)
char *s1;
char *s2;
```

DESCRIPTION

'strpbrk' finds the first occurance of any character in 's2' in the string 's1' and returns a pointer to the found character. If no occurance of any character in 's2' is found, a NULL is returned.

SEE ALSO

strcmp (subs), strcpy (subs), strcat (subs)

NAME

strrchr - find character in string

SYNOPSIS

```
char *strrchr(s,c)
char *s;
char c;
```

DESCRIPTION

'strrchr' finds the first occurance of 'c' in the string 's' starting at the end of 's', working backwards and returns a pointer to the found character. If the character is not found a NULL is returned.

SEE ALSO

strcmp (subs), strcpy (subs), strcat (subs)

NAME

strspn - string segment pattern count

SYNOPSIS

```
strspn(str,pat)
char *str, *pat;
```

DESCRIPTION

'strspn' returns a count of the number of characters in the first string which also occur in the second string, without a break. For example: if str = "abcdefg" and pat = "cabbage", then 'strspn' would return 3, because there was not a 'd' in the second string.

SEE ALSO

NAME

strtok - string, token manipulation

SYNOPSIS

```
char *
strtok(str,toksep)
char *str, *toksep;
```

DESCRIPTION

'strtok' takes as input 'str', a string of tokens; and 'toksep', a string of token separators. It returns a pointer to the first character in 'str' which is not a token separator. The first character following the token (a token separator) is nulled out. On subsequent calls to this routine a NULL (long 0) passed instead of 'str' will cause it to continue parsing the original string. A NULL will be returned if no more tokens remain. Makes use of a static character pointer.

SEE ALSO

strcmp (subs), strcpy (subs)

NAME
tod - time of day

SYNOPSIS
```
long tod();
```

DESCRIPTION
'tod' returns the current time of day.

SEE ALSO
time (sys)

NAME

ttyflush - flush input buffers

SYNOPSIS

```
ttyflush(fd)
int fd;
```

DESCRIPTION

'ttyflush' flushes the input buffer associated with the file descriptor 'fd'. Typically used to flush standard input before requesting input from the terminal.

SEE ALSO

ioctl(sys)

NAME

ttyn - tty name

SYNOPSIS

```
ttyn(fd)
int fd;
```

DESCRIPTION

'ttyn' returns the number of the tty associated with the specified open file descriptor. For example, tty15 would return the number 15.

-1 is returned if the file descriptor does not point to an open file or to a terminal.

SEE ALSO

tty (cmnd)

NAME
ttyname - tty name string

SYNOPSIS
```
char *
ttyname(fd)
int fd;
```

DESCRIPTION
'ttyname' returns a null terminated string that is the name of the tty device associated with the given filedescriptor. NULL is returned if the filedescriptor is not a device or is not in the directory /dev.

FILES
/dev/tty*

DIAGNOSTICS

SEE ALSO
ttyn(subs), isatty(subs)

NAME

twait - wait for ended or stopped task

SYNOPSIS

```
twait(info,flag);
char *info;
int flag;
```

DESCRIPTION

'twait' allows a parent task to wait for the termination or stopping of a child task. If there are no children, an error is returned. If there is currently a stopped or terminated child, the status and process id of the child is returned. If there are no stopped or terminated children, the task is put to sleep waiting for a child to stop or terminate. The task is awoken upon a child stopping or terminating or upon a signal being sent to the calling task.

DIAGNOSTICS

Returns a -1 if an error occurs.

SEE ALSO

wait (sys)

NAME
cgetc - get a single character

SYNOPSIS
```
cgetc(fd);
int fd;
```

DESCRIPTION
Cgetc returns a single character. Fd is a file descriptor for the file from which the character is to be gotten.

FILES

DIAGNOSTICS
A zero is returned on end of file.

SEE ALSO
cputc(subs), getc(subs), putchar(subs), getchar(subs)

NAME

cputc - put a single character

SYNOPSIS

```
cputc(ch,fd);
char ch;
int fd;
```

DESCRIPTION

Cputc puts the single character ch into the file pointed to by the file descriptor fd and the current end of file. Fd is a file descriptor for the file from which the character is to be gotten.

FILES

SEE ALSO

cgetc(subs), putc(subs)

NAME

fcreat - create a file

SYNOPSIS

```
fcreat(fname,ibuf);
char *fname;
struct iob *ibuf;
```

DESCRIPTION

Fcreat creates a file with the name fname and associates an io buffer ibuf with it. The file is created with mode 0666 (see 'chmod (cmnd)' for more on file modes). The buffer is of the following form.

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer definition.

SEE ALSO

fopen(subs), fclose(subs)

DIAGNOSTICS

Fcreat returns -1 if the file can not be created.

BUGS

There is no way to specify the mode to create the file.

NAME

fflush - flushes an io buffer

SYNOPSIS

```
fflush(ibuf);
struct iob *ibuf;
```

DESCRIPTION

Fflush takes everything in the io buffer and writes it out to the associated file. The buffer is of the following form.

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer definition.

SEE ALSO

flush(subs)

DIAGNOSTICS

Flush returns a 0 if successful, and a -1 on a failure.

NAME
flush - output buffered data

SYNOPSIS
flush()

DESCRIPTION
Flush causes all unwritten data from the standard output file 'fout' to be written out.

DIAGNOSTICS
Flush returns a 0 on success and a -1 on failure.

SEE ALSO
fflush(subs)

NAME

fopen - open a file.

SYNOPSIS

```
fopen(fname,ibuf);
char *fname;
struct iob *ibuf;
```

DESCRIPTION

Fopen opens an already existing file fname and associates an io buffer ibuf with it. The buffer is of the following form.

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer definition.

SEE ALSO

fcreat(subs), fclose(subs)

DIAGNOSTICS

Fopen returns -1 if the file can not be opened.

NAME

getc - get a character from an io buffer

SYNOPSIS

```
getc(ibuf)
struct iob *ibuf;
```

DESCRIPTION

Getc gets a character out of the input stream being maintained by the io buffer ibuf. It returns the character. The buffer is of the following form:

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer defines.

SEE ALSO

putc(subs), ungetc(subs), getchar(subs)

NAME

getchar - get a character

SYNOPSIS

```
getchar();
extern struct iobuf fin;
```

DESCRIPTION

This routine gets a character from the file associated with the external file descriptor 'fin'. Unless otherwise set by the user, 'fin' is set to the standard input file descriptor 0.

If the file descriptor for 'fin' is zero (0), the reads will be unbuffered. This is suitable for keyboard input. To use 'getchar' with file I/O, the file should be opened and the file descriptor assigned to 'fin'.

DIAGNOSTICS

'getchar' returns 0 on reaching end of file.

SEE ALSO

putchar(v6io), cgetc(subs)

NAME

getline - get a line of buffered input

SYNOPSIS

```
getline(iobuf,line,len)
struct iob *iobuf;
char *line;
int len;
```

DESCRIPTION

'getline' returns a line of input from the buffered input stream being maintained by the structure iobuf. The iobuf structure is as follows:

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

Characters are read until a carriage return or until 'len' bytes are read. The actual number of bytes read are returned, or zero for end-of-file. The carriage return is included in the line, which will be null terminated.

The REGULUS escape convention is observed, so that more than 1 line may be read if the newline is escaped with a '\'. The backslant '\' and the newline will both become part of the resulting line.

FILES

/usr/include/getc.h io buffer defines.

SEE ALSO

putc (subs)
ungetc (subs)
cgetc (subs)
cputc (subs)
getchar (subs)
putchar (subs)

NAME

getw - get a word from an io buffer

SYNOPSIS

```
getw(ibuf)
struct iob *ibuf;
```

DESCRIPTION

Getw gets a word out of the input stream being maintained by the io buffer ibuf. It returns the word. The buffer is of the following form:

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer defines.

SEE ALSO

getc(subs), putw(subs)

VERSION 6 I/O

This section describes I/O subroutines that are compatible with UNIX Version 6. Although these routines are obsolete, they are included here to provide compatability the oldest version of UNIX. The binary versions of the subroutines reside in the file called lib6.a

The Version 6 I/O subroutines are described in this section in the following sequence:

NAME	Name of the subroutine
SYNOPSIS	Syntax for calling the subroutine.
DESCRIPTION	Briefly describes the results or purpose of the subroutine.
FILES	Lists the file names that are built into the program.
DIAGNOSTICS	Describes routine for diagnosing programming errors.
SEE ALSO	Gives pointers to related information.
BUGS	Lists known program defects of errors

NAME

putc - put a character into an io buffer

SYNOPSIS

```
putc(ch,ibuf)
char ch;
struct iob *ibuf;
```

DESCRIPTION

Putc puts the character ch into the output stream being maintained by the io buffer ibuf. The buffer is of the following form.

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer definition.

SEE ALSO

getc(subs), ungetc(subs), cgetc(subs), cputc(subs), getchar(subs) putchar(subs)

DIAGNOSTICS

Putc returns the character ch if successful, otherwise it returns a -1.

NAME

putchar - write out a character

SYNOPSIS

```
putchar(ch);
char ch;
extern struct iobuf fout;
```

DESCRIPTION

'putchar' is used to write a character to the file associated with the external structure 'fout'. Unless changed by the user, this file points to the standard output file descriptor 1.

If the file descriptor is 1, the output is unbuffered. This is suitable to terminal I/O. For file I/O, the file should be open and the file descriptor assigned to 'fout'.

SEE ALSO

getchar(v6io), getc(subs), ungetc(subs), cgetc(subs), cputc(subs)

DIAGNOSTICS

'putchar' returns the character output if it was successful otherwise it returns a -1.

NAME

putw - put a word into an io buffer

SYNOPSIS

```
putc(w,ibuf)
int w;
struct iob *ibuf;
```

DESCRIPTION

Putw puts the word w into the output stream being maintained by the io buffer ibuf. The buffer is of the following form:

```
struct iob {
        int fd;   /* file descriptor */
        int cc;   /* char cnt        */
        char *cp; /* ptr to next ch */
        char cbuf[512];
};
```

FILES

/usr/include/getc.h io buffer definition.

SEE ALSO

getw(subs), putc(subs)

DIAGNOSTICS

Putw returns the word w if successful, otherwise it returns a -1.

NAME

scanf - formatted input

SYNOPSIS

scanf (flist); char *flist;

DESCRIPTION

Scanf reads from standard input interprets the input according to the format statements and stores it accordingly. There are two types of items in the format string: formatting statements and locations in which to store the input. The command would look something like: scanf("...",arg1,arg2...). The location between the quotes would contain the format statements.

The format string may contain spaces, tabs and new lines which will be ignored, percent signs '%', and conversion characters. The conversion characters must be preceded by a percent sign. An optional maximum field width may be inserted between the two, or an asterisk '*' specifying that this item will be ignored. No other characters are allowed.

The conversion characters and their expected storage types are as follows:

'c' The next character will be stored in a character pointer. It will accept blanks, tabs or carriage returns as legitimate input. If you wish to get the first character which is not a blank, tab or carriage return specify a width field of '1'.

's' The next input will be a character string. It should be stored in a character pointer which has been allocated sufficient space for the expected string.

'd' The next input will be a decimal numeric. It should be stored in an integer pointer.

'o' The next input will be interpreted as an octal numeric. It should be stored in an integer pointer.

'x' The next input will be interpreted as a hexidecimal numeric. It should be stored in an integer pointer.

'h' The next input will be a short integer. It should be stored in a pointer to a short integer.

'f' The next input will be a floating point number. It should be stored in a pointer to float. It will be of the form: an optional minus sign, followed by digits, followed by a decimal point, followed by more digits and an optional exponent field.

'e' Same as 'f'.

Format descriptors 'd', 'o', 'x', and 'u' may optionally be preceeded by a 'l' character to specify that the argument to be interpreted is a long rather than an integer.

Format descriptors 'd', 'o', 'x', and 'u' may optionally be preceeded by a 'l' character to specify that the argument to be printed is a long rather than an integer.

SEE ALSO

printf (subs)

BUGS

The 'e' and 'f' options are not yet implemented.

NAME

ungetc - return a character to an io buffer

SYNOPSIS

```
ungetc(ch,fd);
char ch;
int fd;
```

DESCRIPTION

Ungetc puts the character ch back into the file fd and backs up the file location to point at ch.

DIAGNOSTICS

If too many characters are returned without a subsequent read, the buffer used for this may fill up. An error will be printed and 'ungetc' will return -1.

SEE ALSO

getc(subs), putc(subs)

STANDARD I/O (VERSION 7 UNIX)

This section describes the standard I/O (Version 7) subroutines. Their binary versions reside in the file called lib7.a.

The standard I/O subroutines are described in this section in the following sequence:

NAME	Name of the standard I/O subroutine
SYNOPSIS	Syntax for calling the subroutine.
DESCRIPTION	Briefly describes the results or purpose of the subroutine.
FILES	Lists the file names that are built into the program.
DIAGNOSTICS	Describes routine for diagnosing programming errors.
SEE ALSO	Gives pointers to related information.
BUGS	Lists known program defects or errors.

NAME
clearerr - clear error status

SYNOPSIS
```
#include <stdio.h>

clearerr(stream)
FILE *stream;
```

DESCRIPTION
'clearerr' clears the error status associated with the specified stream. This function is implemented as a macro, it can not be redeclared.

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
feof (stdio)
ferror (stdio)

NAME

isspace, isupper, islower, isalpha, isdigit, isascii, isalnum, ishex, iscntrl, isprint, ispunct, toupper, tolower, toascii - character classifications

SYNOPSIS

```
#include <ctype.h>

isspace(c)
char c;
 ...
```

DESCRIPTION

These routines are all macro definitions which take a single character argument and return an integer.

isspace(c)	true if c is a space, tab, carriage return, newline, or formfeed character.
isupper(c)	true if c is an upper case letter.
islower(c)	true if c is a lower case letter.
isalpha(c)	true if c is an alphabetic character.
isdigit(c)	true if c is a decimal character.
ishex(c)	true if c is an hexidecimal character.
isascii(c)	true if c is in the ascii character range.
isalnum(c)	true if c is an alphanumeric character.
iscntrl(c)	true if c is an ascii control character.
isprint(c)	true if c is a printable character.
toupper(c)	returns the specified characters upper case equivalent.
tolower(c)	returns the specified characters lower case equivalent.
toascii(c)	all non ascii bits will be masked and the resulting character returned.

The return of a zero signifies a false result, a non-zero return signifies true.

FILES

/usr/include/ctype.h - macro definitions

SEE ALSO

NAME
fclose - close a stream

SYNOPSIS
```
#include <stdio.h>

fclose(stream)
FILE *stream;
```

DESCRIPTION
'fclose' closes the file which is associated with the specified stream. 'fclose' also flushes the buffers which had been allocated to the stream.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
returns a -1 if the specified stream is not currently open.

SEE ALSO
fopen (stdio)
fflush (stdio)

NAME

fdopen - open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *
fdopen (fd, mode)
int fd;
char *mode;
```

DESCRIPTION

'fdopen' associates a stream with a file descriptor which was previously attained through another method, such as 'open (sys)' or 'pipe (sys)'. The file access modes are defined by 'mode':

"w"	create for writing.
"a"	open for writing, or if non-existent create.
"r"	open for reading.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns 0 if there are no available streams.

SEE ALSO

fopen (stdio)
freopen (stdio)
close (sys)
open (sys)
pipe (sys)

NAME

feof - get eof status for a stream

SYNOPSIS

```
#include <stdio.h>

feof(stream)
FILE *stream;
```

DESCRIPTION

'feof' returns non-zero if end-of-file on the specified stream has been reached. Otherwise, zero is returned. This function is implemented as a macro and can not be redeclared.

FILES

/usr/include/stdio.h - macro definition

SEE ALSO

ferror (stdio)
fileno (stdio)
clearerr (stdio)

NAME

ferror - report stream error status

SYNOPSIS

```
#include <stdio.h>

ferror(stream)
FILE *stream;
```

DESCRIPTION

'ferror' returns non zero if an error has occurred while reading or writing to the specified stream, zero otherwise. The error can only be cleared by closing the file or by calling 'clearerr (stdio)'. This function is implemented as a macro and therefore can not be redeclared.

FILES

/usr/include/stdio.h - macro definition

SEE ALSO

feof (stdio)
clearerr (stdio)
fileno (stdio)

NAME

fflush - flush a stream

SYNOPSIS

```
#include <stdio.h>

fflush(stream)
FILE *stream;
```

DESCRIPTION

'fflush' flushes the buffer associated with a stream. If the stream passed as an argument to 'fflush' is an output stream the buffers associated with it will be written out. If the stream passed to this routine is an input buffer the buffer will be flushed. 'fflush' does not close the buffers.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

returns a -1 if the buffers can not be flushed or if the specified stream has not been opened.

SEE ALSO

fclose(stdio), setbuf(stdio)

NAME
fgetc - get a character

SYNOPSIS
```
#include <stdio.h>

fgetc(stream)
FILE *stream;
```

DESCRIPTION
'fgetc' is the function version of the macro 'getc (stdio)'. They work exactly the same. This can be used to cut object code size. A -1 is returned upon reaching end of file, defined as 'EOF'.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
EOF (-1) is returned on end-of-file.

SEE ALSO
getc (stdio)
getchar (stdio)
fputc (stdio)

NAME

fgets - get a string

SYNOPSIS

```
#include <stdio.h>

char *
fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

DESCRIPTION

'fgets' reads the next string of characters terminated by a carriage return or at most n-1 characters from the buffer associated with stream into the string pointed to by s. The carriage return is retained and followed by a null. The parameter s is returned.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns NULL on end of file being reached.

SEE ALSO

fputs (stdio)
gets (stdio)

NAME
fileno - get stream file descriptor

SYNOPSIS
```
#include <stdio.h>

fileno(stream)
FILE *stream;
```

DESCRIPTION
'fileno' returns the file descriptor which is currently associated with the specified stream.

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
fopen (stdio)

NAME
fopen - open a stream

SYNOPSIS
```
#include <stdio.h>

FILE *
fopen (fname, mode)
char *fname, *mode;
```

DESCRIPTION
'fopen' opens the specified file according to the specified mode and associates a stream with it. The file modes are:

"w"	create for writing.
"a"	open for writing, or if non-existant create.
"r"	open for reading.
"r+"	open for update (reading and writing)
"w+"	create for update
"a+"	append; open or create for update at end of file

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns 0 if the file could not be accessed.

SEE ALSO
fdopen (stdio), freopen (stdio), fclose (stdio)

NAME

fprintf - formatted print

SYNOPSIS

```
#include <stdio.h>

fprintf(stream,plist)
FILE *stream;
char *plist;

sprintf(s,plist)
char *s, *plist;
```

DESCRIPTION

'fprintf' takes the input string and interprets various formatting commands and outputs the results to an output stream. 'sprintf' directs the output into a null terminated string.

There are two types of items in the argument string: characters which are copied literally and format statements which work on strings, characters and numerics. The string takes the form of a literal string with embedded format statements and the various arguments on which the format statements act: printf("...",arg1,arg2...). All format statements are preceded by a percent sign and terminated by one of the conversion characters. Between the percent sign and the conversion characters are optional symbols which adjust the standard formats.

The format statement consists of a contiguous group of characters. A minus sign '-' may follow the percent sign to designate that the item is to be left justified. A decimal number designating the field width may optionally be specified. If the item is larger than the specified width it will be printed as is. If the item is smaller, the item will be padded with spaces. If the first digit of the specified width is a '0' then the item will be padded with zero's instead of spaces. If the item is a string or a floating point number a second field preceded by a period '.' may be specified. The second width field specifies the number of digits to the right of the decimal point in the case of a floating point and the maximum number of characters to be printed in the case of a string.

The last item of a format statement describes the item we are formatting. The options are 'c', 'd', 'e', 'f', 'u', 'o', 's', and 'x'.

'c' The argument to be printed is a single character.

's' The argument is a string or character pointer. All characters will be printed unless a maximum width field is in force.

'd' The argument is a decimal numeric.

'o' The argument is a octal numeric.

'x' The argument is a hexidecimal numeric.

'u' The argument will be interpreted as an unsigned decimal integer in the range 0 to 65535.

'f' The argument is either a floating point or double precision number. It will be of the form: an optional minus sign, digits, a decimal point, and more digits. If no second width field has been specified there will be six digits to the right of the decimal point.

'e' The argument is to be printed in scientific notation and is either a floating point or double precision number. The format is exactly like the 'f' format except that only one digit is to the right of the decimal point and a two digit exponent is specified after the number.

Format descriptor characters 'd', 'o', 'x', and 'u' may be optionally preceeded by a 'l' character to specify that the argument to be printed is a long rather than a integer.

Any other character following a percent sign will be taken as a literal and will itself be printed. In this way you can print a percent sign or a double quote.

FILES

/usr/include/stdio.h - macro definitions

SEE ALSO

scanf (stdio)

NAME
fputc - output a character

SYNOPSIS
```
#include <stdio.h>

fputc(ch,stream)
char ch;
FILE *stream;
```

DESCRIPTION
'fputc' is the function version of the macro 'putc (stdio)'. They work exactly the same. This may be used in an effort to cut object code size.

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
putc (stdio)
putchar (stdio)
fgetc (stdio)

NAME
fputs - output a string

SYNOPSIS
```
#include <stdio.h>

fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION
'fputs' outputs the null terminated string to the specified stream. The null is not transferred.

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
puts (stdio)
gets (stdio)

NAME
fread - read from stream

SYNOPSIS
```
#include <stdio.h>

fread(ptr,size,n,stream)
char *ptr;
int size, n;
FILE *stream;
```

DESCRIPTION
'fread' reads 'n' input records of the specified 'size' into the buffer pointed to by 'ptr' from the specified input stream. 'fread' returns the number of records it read.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
A zero is returned if the associated stream was not open for reading.

SEE ALSO
fwrite (stdio)
fopen (stdio)

NAME

freopen - open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *
freopen (fname, mode, stream)
char *fname, *mode;
FILE *stream;
```

DESCRIPTION

'freopen' opens the specified file according to the specified mode and substitutes it for the file currently associated with the specified stream. The file modes are:

"w"	create for writing.
"a"	open for writing, or if non-existent create.
"r"	open for reading.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns 0 if the file could not be accessed. Returns -1 if the associated stream has not already been opened for reading or writing.

SEE ALSO

fopen (stdio)
fdopen (stdio)
close (stdio)

NAME

fseek - reposition a stream

SYNOPSIS

```
#include <stdio.h>

fseek(stream,offset,origin)
FILE *stream;
long offset;
int origin;
```

DESCRIPTION

'fseek' repositions the location of the file pointer to the location 'offset' bytes from the beginning, current location or end of the file as specified by an origin of 0, 1, or 2 respectively. The buffer is flushed before the file pointer is changed.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns a -1 if the seek is unsuccessful, or if the stream has not been opened.

SEE ALSO

rewind (stdio)
ftell (stdio)
lseek (stdio)

NAME
ftell - file pointer relative offset

SYNOPSIS
```
#include <stdio.h>

long
ftell (stream)
FILE *stream;
```

DESCRIPTION
'ftell' returns the file pointers current byte offset from the beginning of the stream.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns a -1 if the specified stream has not been opened for reading or writing.

SEE ALSO
fseek (stdio), rewind (stdio), lseek (sys)

NAME
fwrite - write to a stream

SYNOPSIS
```
#include <stdio.h>

fwrite(ptr,size,n,stream)
char *ptr;
int size,n;
FILE *stream;
```

DESCRIPTION
'fwrite' flushes the output buffers, then writes 'n' records of the specified 'size' starting at the location pointed to by 'ptr' to the specified stream. 'fwrite' returns the number of records which were successfully written.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns a 0 if the specified stream was not opened for writing.

SEE ALSO
fread (stdio), fopen (stdio)

NAME
getc - get a character

SYNOPSIS
```
#include <stdio.h>

getc(stream)
FILE *stream;
```

DESCRIPTION
'getc' returns the next character in the buffer associated with the specified stream. This is implemented as a macro and therefore can not be redeclared.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
EOF (-1) is returned on end of file or error.

SEE ALSO
fgetc (stdio)
getchar (stdio)
putc (stdio)

NAME
getchar - get a character

SYNOPSIS
#include <stdio.h>

getchar()

DESCRIPTION
'getchar' returns the next character in the standard input stream. 'getchar' is implemented as a macro.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
EOF (-1) is returned on end of file or error.

SEE ALSO
getc (stdio)
putchar (stdio)

NAME

getline - get a line of buffered input

SYNOPSIS

```
#include <stdio.h>

getline(stream,line,len)
FILE *stream;
char *line;
int len;
```

DESCRIPTION

'getline' returns a line of input from the buffered input stream. Characters are read until a carriage return or until 'len' bytes are read. The actual number of bytes read are returned, or zero for end-of-file. The carriage return is included in the line, which will be null terminated.

The REGULUS escape convention is observed, so that more than 1 line may be read if the newline is escaped with a '\'. The backslant '\' and the newline will both become part of the resulting line.

FILES

/usr/include/stdio.h macro definitions.

SEE ALSO

putc (stdio)
ungetc (stdio)
cgetc (stdio)
cputc (stdio)
getchar (stdio)
putchar (stdio)

NAME
gets - get a string

SYNOPSIS
```
#include <stdio.h>

char *
gets (s)
char *s;
```

DESCRIPTION
'gets' reads the next string of characters terminated by a carriage return from the standard input stream into the string pointed to by s. The carriage return is replaced by a null. The parameter s is returned.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns NULL on end of file being reached, or on error.

SEE ALSO
fgets (stdio)
puts (stdio)

NAME
getw - get a word

SYNOPSIS
```
#include <stdio.h>

getw(stream)
FILE *stream;
```

DESCRIPTION
'getw' returns the next word from the buffer associated with the specified stream.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns an EOF on end of file or on error.

SEE ALSO
getc (stdio)
fgetc (stdio)

NAME
pclose - terminate i/o to/from a pipe

SYNOPSIS
```
#include <stdio.h>

pclose(stream)
FILE *stream;
```

DESCRIPTION
'pclose' waits for the child process to terminate, then closes the stream associated with it. It returns the exit status of the child process.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
A -1 is returned if a pclose is attempted on a stream that was not opened with 'popen'.

SEE ALSO
popen (stdio)
pipe (sys)

PROGRAM

USAGE

```
perror(str)
char *str;
```

FUNCTION

'perror' prints an error message according to the value currently in errno. The string argument is generally a pointer to the source of the error source (eg. perror("open");).

SEE ALSO

BUGS

NAME

popen - initiate i/o to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *
popen(command,mode)
char *command, *mode;
```

DESCRIPTION

'popen' is used to create a pipe between the host process and its child. A mode of "r" institutes a pipe for reading from the child process and a mode of "w" is used for writing to the child process and associates a stream with it. 'popen' attempts to execute the "command" and returns a -1 if it is unable to do so, otherwise it returns a pointer to the pipe stream.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

A -1 is returned if an improper mode is specified.

SEE ALSO

pclose (stdio)
pipe (sys)

NAME
putc - output a character

SYNOPSIS
```
#include <stdio.h>

putc(c,stream)
char ch;
FILE *stream;
```

DESCRIPTION
'putc' outputs the specified character to the output buffer associated with the specified stream. 'putc' is implemented as a macro.

DIAGNOSTICS
'putc' returns EOF (-1) on error (for example, trying to put a character on a read-only file stream).

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
fputc (stdio)
putchar (stdio)
getc (stdio)

NAME

putchar - output a character

SYNOPSIS

```
#include <stdio.h>

putchar(ch)
char ch;
```

DESCRIPTION

'putchar' outputs the specified character ch to the standard output stream. 'putchar' is implemented as a macro and therefore can not be redeclared.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns EOF (-1) on error.

SEE ALSO

putc (stdio)
getchar (stdio)

NAME
puts - output a string

SYNOPSIS
```
#include <stdio.h>

puts(s)
char *s;
```

DESCRIPTION
'puts' outputs the null terminated string s to the standard output device. A carriage return is appended to the string, and the null is not output.

FILES
/usr/include/stdio.h - macro definitions

SEE ALSO
fputs (stdio)
gets (stdio)

NAME

putw - output a word

SYNOPSIS

```
#include <stdio.h>

putw (w, stream)
int w;
FILE *stream;
```

DESCRIPTION

'putw' outputs the specified word into the file associated with the specified stream. 'putw' returns the word it outputs.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

'putw' returns EOF (-1) on error. 'ferror' (subs) should be used to detect an error, since EOF (-1) is a valid integer that may have been output.

SEE ALSO

getw (stdio)
putc (stdio)
fputc (stdio)
ferror (stdio)

NAME
rewind - reposition a stream

SYNOPSIS
```
#include <stdio.h>

rewind(stream)
FILE *stream;
```

DESCRIPTION
'rewind' repositions the file pointer associated with the specified stream to the beginning of the stream. 'rewind(stream)' is identical to 'fseek(stream,(long) 0, 0)'.

FILES
/usr/include/stdio.h - macro definitions

DIAGNOSTICS
Returns -1 if the rewind was unsuccessful.

SEE ALSO
fseek (stdio)
ftell (stdio)
lseek (stdio)

NAME

scanf - formatted input

SYNOPSIS

```
#include <stdio.h>

scanf (flist);
char *flist;

fscanf(stream,flist)
FILE *stream;
char *flist;

sscanf(s,flist)
char *s, *flist;
```

DESCRIPTION

'scanf' reads from standard input interprets the input according to the format statements and stores it accordingly. 'fscanf' reads from the specified input stream, and 'sscanf' reads from the specified string. There are two types of items in the format string: formatting statements and locations in which to store the input. The call would look something like: scanf("...",arg1,arg2...). The location between the quotes would contain the format statements.

The format string may contain spaces, tabs and new lines which will be ignored, percent signs '%', and conversion characters. The conversion characters must be preceded by a percent sign. An optional maximum field width may be inserted between the two, or an asterisk '*' specifying that this item will be ignored. No other characters are allowed.

The conversion characters and their expected storage types are as follows:

'c' The next character will be stored in a character pointer. It will accept blanks, tabs or carriage returns as legitimate input. If you wish to get the first character which is not a blank, tab or carriage return specify a width field of '1'.

's' The next input will be a character string. It should be stored in a character pointer which has been allocated sufficient space for the expected string.

'd' The next input will be a decimal numeric. It should be stored in an integer pointer.

'o' The next input will be interpreted as an octal numeric. It should be stored in an integer pointer.

'x' The next input will be interpreted as a hexidecimal

numeric. It should be stored in an integer pointer.

'h' The next input will be a short integer. It should be stored in a pointer to a short integer.

'f' The next input will be a floating point number. It should be stored in a pointer to float. It will be of the form: an optional minus sign, followed by digits, followed by a decimal point, followed by more digits and an optional exponent field.

'e' Same as 'f'.

Format descriptors 'd', 'o', 'x', and 'u' may optionally be preceeded by a 'l' character to specify that the argument to be printed is a long rather than an integer.

FILES
/usr/include/stdio.h - macro definitions.

SEE ALSO
fprintf (stdio)

BUGS
The 'e' and 'f' options are not yet implemented.

NAME

setbuf - assign a stream buffer

SYNOPSIS

```
#include <stdio.h>

setbuf(stream,buf)
FILE *stream;
char *buf;
```

DESCRIPTION

'setbuf' allows the user to allocate the buffer for an opened stream previous to the first read or write action on it takes place. If buf is a NULL pointer the input and output on the designated stream will be unbuffered. If buf is a long -1 (i.e. (char *)-1), the designated stream will be buffered using a system allocated buffer. Otherwise the buffer must be allocated BUFSIZE bytes by the user, where BUFSIZE is a predeclared constant in the 'stdio' library package.

```
char buf[BUFSIZE];
```

Buffers are generally allocated the first time reading or writing is attempted on a stream. The 'stdin' stream is buffered and the 'stdout' and 'stderr' streams are generally unbuffered.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

If the stream is not open a -1 will be returned.

SEE ALSO

fopen (stdio)

NAME

stdio - standard input/output package

SYNOPSIS

#include <stdio.h>

DESCRIPTION

The REGULUS standard i/o package allows for accessing files and performing input and output functions in a simple and straightforward way. There are two macro defined functions which most of the other routines use. These are 'getc (stdio)' and 'putc (stdio)'. 'getchar (stdio)', 'putchar (stdio)', 'feof (stdio)', 'ferror (stdio)', 'clearerr (stdio)' and 'fileno (stdio)' are also implemented as macros. Sequential buffered transactions are the most efficient, but files may be randomly accessed using the 'fseek (stdio)', 'rewind (stdio)' and 'ftell (stdio)' functions.

The file 'stdio.h' defines the macros and declares the variables necessary for use by the library functions. Beware of incrementing pointers inside of the macro's argument list, the results are not guaranteed to be what you expect.

Any program which makes use of these routines must include this file. A line of the form "#include <stdio.h>" is used in C.

These functions work on a standardly defined buffer stream called a FILE. There are normally three open streams associated with a task. These are the standard input ('stdin'), standard output ('stdout') and standard error ('stderr').

SEE ALSO

clearerr(stdio), fclose(stdio), fdopen(stdio), feof(stdio), ferror(stdio), fflush(stdio), fgetc(stdio), fgets(stdio), fileno(stdio), fopen(stdio), fprintf(stdio), fputc(stdio), fputs(stdio), fread(stdio), freopen(stdio), fscanf(stdio), fseek(stdio), ftell(stdio), fwrite(stdio), getc(stdio), getchar(stdio), gets(stdio), getw(stdio), pclose(stdio), popen(stdio), putc(stdio), putchar(stdio), puts(stdio), putw(stdio), rewind(stdio), setbuf(stdio), ungetc(stdio)

DIAGNOSTICS

A -1 is generally returned upon end of file or an attempt to read from or write to a file which is not opened for such accesses.

NAME

ungetc - put character back into a stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
char c;
FILE *stream;
```

DESCRIPTION

'ungetc' returns a character to a stream. If the stream is buffered and a character has been read from it, at most one character may be pushed back. Pushing back the end of file character is not allowed. If the 'ungetc' is successful, the character returned to the buffer will be returned.

FILES

/usr/include/stdio.h - macro definitions

DIAGNOSTICS

Returns EOF (-1) if unsuccessful.

SEE ALSO

getc (stdio)
fgetc (stdio)

FILES

This section describes the format of various system files and the general file structure of the system itself.

The C-language structure of each of these formats is usually found in the directory

/usr/include

with an example of a specified file name being

/usr/include/ar.h

which describes the format of archives.

Files are described in this section in the following sequence:

NAME Name of the file structure.

DESCRIPTION Briefly describes the file structure.

SEE ALSO Gives pointers to related information.

NAME

a.out - 68000 loader output

DESCRIPTION

'a.out' is the output file of the loader 'ld (cmnd)'. The loader makes 'a.out' executable if there were no errors and no unresolved external references. The *.o relocatable object files which are produced by the assembler 'as (cmnd)' have this same format.

This file has four sections: a header, the program and data text, a symbol table, and relocation bits (in that order). The last two may be empty if the program was loaded with the '-s' option of 'ld (cmnd)'.

There are two formats of the header. The first header format starts with the identifying number 601A hex and contains 14 words in the following format:

```
struct hdr {
    int magic;          //a.out magic number = 601A hex
    long tsize;         //# bytes in program text segment
    long dsize;         //# bytes in program data segment
    long bsize;         //# bytes in program bss  segment
    long ssize;         //# bytes in symbol table
    long stksize;       //initial stack size
    long entry;         //entry point
    int rlbflg;         //relocation bits suppressed flag
 };
```

The second header format is used when noncontiguous text, data, and bss segments are specified to the loader using the -T, -D, and -B flags. This header begins with the identifying number 601B hex and contains 18 words in the following format:

```
struct hdr {
    int magic;          //a.out magic number = 601B hex
    long tsize;         //# bytes in program text segment
    long dsize;         //# bytes in program data segment
    long bsize;         //# bytes in program bss  segment
    long ssize;         //# bytes in symbol table
    long stksize;       //initial stack size
    long entry;         //entry point--address of text segmen
    int rlbflg;         //relocation bits suppressed flag
    long dstart;        //address of data segment
    long bstart;        //address of bss segment
 };
```

The sizes of each segment are in bytes but are even. The size of the header is not included in any of the other sizes.

When a file produced by the loader is loaded into memory for execution, three logical segments are set up: the text

segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at the address contained in "entry" in the core image; the header is not loaded. If the magic number (word 0) is 601A hex, it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is 601C hex, it indicates that the text segment is to be write protected and shared and that the beginning of the data segment is at the first 2K byte boundary beyond the text segment. If the magic number is 601D hex, it indicates that the text segment and the data segment are both loaded with virtual addresses beginning at 0 (I and D split). If the magic number is 601E hex, it indicates that the text segment is to be write protected and shared and that the beginning of the data segment is at the first 4K byte boundary beyond the text segment.

The symbol table consists of 7-word entries. The first four words contain the ASCII name of the symbol, null-padded. The next word is a flag indicating the type of symbol. The following values are possible:

0100000	defined
0040000	equated
0020000	global - entry or external
0010000	equated register
0004000	external reference
0002000	DATA based relocatable
0001000	TEXT based relocatable
0000400	BSS based relocatable

These values are bit flags which may be or'ed together to indicate completely a symbol type such as 0122000 for a defined global data segment relocatable symbol.

The last two words (long) of a symbol table entry contain the value of the symbol.

If the 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.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation bits for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the loader and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the "suppress relocation" flag in the header is on.

Bits 2-0 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

00	indicates the reference is absolute
01	indicates the reference is data segment relocatable
02	indicates the reference is text segment relocatable
03	indicates the reference is bss segment relocatable
04	indicates the reference is to an undefined external sy
05	indicates the reference is to the upper word of a lon
07	indicates the absolute first word of an instruction

The remainder of the relocation word (bits 15-3) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as (cmnd), ld (cmnd), strip (cmnd)

NAME

ar - archive format

DESCRIPTION

The first word in an archive file is a magic number which defines the type of archive file. Magic number (0177545) archives are maintained on REGULUS. Each constituent file in an archive file is preceeded by a file header. The structure of the file header is:

```
struct arhdr {
    char a_fname[14];  // file name
    long a_modti;      // last modified time
    char a_userid;     // user id (uid)
    char a_gid;        // group id (gid)
    int  a_fimode;     // mode word
    long a_fsize;      // file size
};
```

SEE ALSO

ar (cmnd), ld (cmnd)

NAME

c.out - 68000 loader output

DESCRIPTION

'c.out' is the output file of the loader 'ld (cmnd)'. The loader makes 'c.out' executable if there were no errors and no unresolved external references. The *.o relocatable object files which are produced by the assembler 'as (cmnd)' have this same format.

This file has four sections: a header, the program and data text, a symbol table, and relocation bits (in that order). The last two may be empty if the program was loaded with the '-s' option of 'ld (cmnd)'.

There are two formats of the header. The first header format starts with the identifying number 601A hex and contains 14 words in the following format:

```
struct hdr {
    int  magic;  //c.out  magic number = 601A
hex
    long tsize; //#  bytes  in  program  text
segment
    long  dsize;  //#  bytes  in program data
segment
    long bsize;  //#  bytes  in  program  bss
segment
    long ssize; //# bytes in symbol table
    long stksize; //initial stack size
    long entry; //entry point
    int  rlbflg; //relocation bits suppressed
flag
 };
```

The second header format is used when noncontiguous text, data, and bss segments are specified to the loader using the -T, -D, and -B flags. This header begins with the identifying number 601B hex and contains 18 words in the following format:

```
struct hdr {
    int magic;          //c.out magic number = 601B hex
    long tsize;         //# bytes in program text segment
    long dsize;         //# bytes in program data segment
    long bsize;         //# bytes in program bss  segment
    long ssize;         //# bytes in symbol table
    long stksize;       //initial stack size
    long entry;         //entry point--address of text segm
    int rlbflg;         //relocation bits suppressed flag
    long dstart;        //address of data segment
```

```
    long bstart;        //address of bss segment
};
```

The sizes of each segment are in bytes but are even. The size of the header is not included in any of the other sizes.

When a file produced by the loader is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at the address contained in "entry" in the core image; the header is not loaded. If the magic number (word 0) is 601A hex, it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment.

The symbol table consists of 7-word entries. The first four words contain the ASCII name of the symbol, null-padded. The next word is a flag indicating the type of symbol. The following values are possible:

0100000	defined
0040000	equated
0020000	global - entry or external
0010000	equated register
0004000	external reference
0002000	DATA based relocatable
0001000	TEXT based relocatable
0000400	BSS based relocatable

These values are bit flags which may be or'ed together to indicate completely a symbol type such as 0122000 for a defined global data segment relocatable symbol.

The last two words (long) of a symbol table entry contain the value of the symbol.

If the 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.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation bits for that word, then the value of

the word as stored in the file is an offset from the associated external symbol. When the file is processed by the loader and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the "suppress relocation" flag in the header is on.

Bits 2-0 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

00	indicates the reference is absolute
01	indicates the reference is data segment relocatable
02	indicates the reference is text segment relocatable
03	indicates the reference is bss segment relocatable
04	indicates the reference is to an undefined external
05	indicates the reference is to the upper word of a lo
07	indicates the absolute first word of an instruction

The remainder of the relocation word (bits 15-3) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as (cmnd), ld (cmnd), strip (cmnd)

NAME

/etc/dtab - filesystem dump record file

FUNCTION

'/etc/dtab' is the record file used by 'dump (maint)' to record the name and last dump date for each filesystem. The filesystem names are stored as null terminated strings. The date is stored as the long word returned by 'time (sys)'.

There is no fixed size for the filesystem names.

SEE ALSO

dump (maint)

NAME

dump - format of filesystem dumpfiles

FUNCTION

The general structure of a filesystem dump made by 'dump (maint)' is a header block, index node list and data blocks. Data blocks are the actual data of the file, preceded by the file index record.

Each reel of a multi-reel dump begins with a header block. This is used to determine if the user is out of sequence.

The following structure defines the header block:

```
struct dheader {
    long h_nindexes; // number of index nodes in dump
    long h_nfiles;   // number of files in dump
    long h_nblocks;  // number of blocks in dump
    long h_incdate;  // incremental dump date
    long h_ddate;    // full dump date
    long h_tblocks;  // total number of blocks in dump
    long h_tsize;    // # of blocks on a tape reel
    int  h_nreels;   // # of reels used in this dump
    int  h_reel;     // reel number of this reel
    long h_start;    // starting block number on this reel
    long h_last;     // last block number on this reel
    char h_label[32]; // reel label
};
```

The index node list has one node for each file (real or special), and the index record numbers are sequential. If the file was not dumped, the dump size is 0. The structure of the index record blocks are as follows:

```
struct dnode {
    long d_index;   // index record number for file
    long d_size;    // file size in blocks + 1 for index
                    // 0=>file not dumped.
};
```

SEE ALSO

dump (maint)
restor (maint)

NAME

filesys - REGULUS file system format

DESCRIPTION

REGULUS File Systems

Each file system storage volume consists of some number of 512 byte blocks, each of which can be randomly accessed. The total file system size is limited to 2**31 blocks, and any individual file is limited in size to 2**31 bytes.

Reserved Blocks

Only the first three blocks of the file system are reserved for system information. Block 0 is unused and is available to contain a bootstrap program or any other desired information.

Block 1 is known as the 'file system superblock' and contains the following information:

```
struct fshead {
    char  label[32];        //pack label
    long  nblocks;          //# blocks in file system
    long  unused;           //not defined
    long  freelst;          //block # of head of free list
    struct direct rootdir; //root directory entry
    long  fsdatim;          //place to remember date and time
};
```

Block 2 contains the index pointers for the root directory of the file system. The remainder of the blocks in the file system are either free blocks, file index records, or file data blocks.

Directories

A directory is treated exactly like a simple data file with the exception that only the system can modify a directory. A directory contains any number of file entries of the format:

```
struct direct {
    char  fname[12];    //file name null padded on right
    long  findblock;    //block # of first index record
};
```

Thus, each file entry in a directory is 16 bytes long allowing 32 such entries in each directory data block. A directory entry is valid if the index record number (findblock) is non-zero. The file name is not necessarily null even in unallocated entries.

Index Records

An index record is the REGULUS file definition structure. Each index record is 256 words long. The first 8 words contain fixed information which may be used to identify and verify index records and the file data blocks. The index record format is:

```
#define NIBPB    122
struct indblk {
    int   i_magic;          //magic # to id index records
    long  i_size;           //file size
    int   i_ownerid;        //file owner id
    char  i_groupid;        //group id of file
    char  i_nlinks;         //file reference count
    long  i_moddate;        //file mod date and time
    long  i_accdate;        //file access date and time
    int   i_flags;          //file type and access mode
    long  i_blkptr[NIBPB]; //block # of data blocks, first
                            //blk ptr contains major/minor
                            //device # for special files
    long  i_nxtiblk;        //next indirect blk or 0
};
```

File Size

Each data block in REGULUS is 512 bytes in length. One index record will therefore allow for a file of 122*512 = 62464 bytes. If a file is larger than 62464 bytes, the last word of the first index record is a pointer to the second index record, etc. Thus, the file system allows for files of arbitrary maximum length. Since a file may not cross a file system boundary and the length of a file system is limited to 32 bits, files are actually limited to something slightly less than 2**32 bytes.

Free List

The free list is a linked list of indirect blocks. Each free list indirect block contains up to 127 block numbers (longs) of free blocks. The last long word in the free list indirect block contains the block number of the next indirect block in the free list chain. This number will be zero at the end of the free list chain. Any block number in the free list that has a value of zero should be ignored since it is impossible for block zero to be free.

SEE ALSO

mount (maint)
umount (maint)
mountfiles (files)
mtab (files)

NAME

/etc/fstab - file system table

FUNCTION

The '/etc/fstab' file is an ascii file composed of the names of the standardly mounted filesystems. Each filesystem name should be terminated by a carriage return. This file is used by 'df (cmnd)'.

SEE ALSO

df (cmnd)

NAME

/etc/group - list of user groups file

DESCRIPTION

/etc/group consists of one line of information per group. There are four colon separated fields:

1) group name
2) the encrypted group passwd
3) group id (gid), in decimal
4) a comma-separated list of users that may belong to this gro

/etc/group is an ascii file. It should have general read permission since commands that use groups needs this information.

SEE ALSO

chgrp (cmnd)
newgrp (cmnd)
passwd (files)

NAME

/etc/mountfiles - mounted filesystem description file

DESCRIPTION

This file describes the default mounting instructions for mountable file systems under REGULUS. The format is as follows:

special file<TAB>root directory<TAB>flags<TAB>comments

The field separator may be either a tab character (^I) or a space. Multiple separators are allowed so that the file may be formatted for easier reading.

The first two items are the special file name, and the directory it is normally mounted on.

The flags are used to instruct mount about how and when to mount that file. The flags are one or more of the following:

a always mount this file (used in 'mount all' command)

r read only. Mount file system read only. This is separate from a hardware read-only switch setting. This flag should be used if there is hardware settings also, otherwise the system will attempt to write the device.

- a null argument. Seperates the comment field from the other fields if no other flag is given.

SEE ALSO

mount (maint)
unmount (maint)
mtab (files)

NAME

/etc/mtab - currently mounted filesystem table

DESCRIPTION

This file attempts to describe the current mounted file configuration. It is maintained by 'mount' and 'unmount' but may not be correct. This file should be interrogated by programs wishing to know about mounted file systems. Each entry is 64 bytes long. The structure is:

```
struct mtab {
    char m_path[32];      // path name of root directory
    char m_special[30];   // device name
    char m_unused;
    char m_rwflag;        // 'r' if device is read-only,
                          // 0 otherwise
};
```

SEE ALSO

mount (maint)
unmount (maint)
mountfile (files)

NAME

/etc/passwd - password files

DESCRIPTION

/etc/passwd consists of one line of information per user. There are six colon separated fields:

1) login name, all lower case
2) encrypted passwd
3) user id (uid), in decimal
4) group id (gid), in decimal
5) initial working (home) directory
6) program to use as shell (/bin/sh is default)

/etc/passwd is an ascii file. It should have general read permission since the passwords are encrypted and everyone needs the information mapping uid to login name.

SEE ALSO

getpw (subs)
cryptpw (subs)
passwd (cmnd)
login (cmnd)
group (files)

NAME

tar - tape archiver

FUNCTION

A 'tar' file is composed of file headers, file data blocks and zero or more blocks of nulls. Each file is preceded by a header block which looks like the following:

```
struct header {
     char fname[100];
     char mode[8];
     char userid[8];
     char grpid[8];
     char bytesize[12];
     char modtime[12];
     char checksum[8];
     char linkflag;
     char linkfname[100];
     char volsize[8];
     char volume[8];
     char record[8];
     char userinfo[64];
}
```

The header block is padded to 512 bytes. Each header is followed by 512 byte blocks of data unless it is linked to another archived file. The end of the tape is padded to the block factor with 512 byte blocks of nulls.

SEE ALSO

tar (cmnd), ar(cmnd)

NAME

ttys - /etc/ttys format

FUNCTION

The file '/etc/ttys' is used to inform the system about terminals (tty's). It is an ascii file of lines of three characters each. The characters are positional and have the following meanings:

CHAR VALUES MEANINGS

1 The first character determines the state of logins for the specified terminal (see character 2).

- 0 Terminal is disabled. Logins are not allowed, and no 'init' will be started for this terminal.
- 1 Terminal is enabled. A copy of 'init' will be created to wait on this terminal to become ready. A terminal is assumed to be ready when an open on the device succeeds.
- ? Terminal is draining. A new copy of 'init' will not be started when the current process running on this terminal is ended. This has the effect of 'draining' the users off the system, since new logins are prohibited when old users log out. Any character other than '0' or '1' is allowed.

2 The second character is the last character of the name of the character-special terminal device refered to by this line of the file. Only alphabetic or single numeric characters are allowed.

3 The last character defines the type of terminal and line this device is connected to. The types are as follows:

- 0 CIT-80 or VT-52 type terminal at 9600 baud.
- 1 CIT-100 or VT-101 type terminal at 9600 baud.
- 2 300 baud dial-in line.
- 3 1200 baud dial-in line.
- 4 1200-300 baud cycling line. The line will be tried first at 1200 baud. If a break is detected, the line is reset to 300 baud and tried. This loop continues until a line is successfully read.

5	300-1200 baud cycling line. This is the same as '4' above, except that 300 baud is tried first.

SEE ALSO
init (maint)
tty (dev)

NAME

/etc/utmp - currently mounted filesystem table

DESCRIPTION

This file attempts to describe who the current users of a REGULUS system are. The file /etc/utmp is maintained by init and getty. There is at most, only one entry for each terminal.

This defines the structure of the utmp file for regulus:

```
#define UTMP_LEN   16  /* number of bytes in a utmp entry */

struct utmp {
   char u_name[8];     /* user name */
   char u_not1;
   char u_tty;         /* last char of tty name */
   long u_timein;      /* login time */
   int  u_not2;
};
```

Each entry is 16 bytes long.

FILES

The above structere can be found in /usr/include/utmp.h

SEE ALSO

who (cmnd)
mountfile (files)

MAINTENANCE

MAINTENANCE

This section describes programs that are used by system administrators and others who deal with maintenance of the file system and of the REGULUS system itself.

NOTE

If you are a new user of the REGULUS system, please be extremely careful in using the maintenance programs. Make sure you understand exactly the results of each command, because it is very easy to destroy important information by using the commands carelessly. For this reason, access to most of the maintenance commands is restricted to the super-user.

PROGRAM

badblk - add badblocks to badblock file

USAGE

badblk [-d special] [-m mount-point] badblock1 ...

FUNCTION

'badblk' enables the user to add additional badblocks to the badblock file thereby avoiding their use by the system. A number of precautions must be exercised before using the 'badblk' program. The system must be up in single-user mode without /etc/update running. The badblocks which are to be given to 'badblk' must be on the free list of the corresponding filesystem.

Once the above conditions are met, 'badblk' may be executed regardless of whether or not any badblocks have previously been specified to the system. In this manner if additional badblocks are discovered they may be added to the badblock file.

If the badblocks are not on the root filesystem, the '-d' option must be used with the special device corresponding to the appropriate filesystem specified. 'badblk' requires a mount-point upon which to mount the specified filesystem. If no mount-point is given with the '-m' option '/mnt' is used as a default.

After 'badblk' has been executed 'fsck -s' must be run on the filesystem containing the badblocks.

'badblk' functions by appending the badblock numbers to the index block '/.BADBLOCK' on the appropriate filesystem. If '/.BADBLOCK' does not exist 'badblk' will create it.

It may be appropriate to remove the entire track containing the badblock from the system. In this case use 'bblist' to obtain the badblock numbers for the entire track.

SEE ALSO

mkfs (maint), bblist (maint)

BUGS

There is currently no way to avoid using badblocks in the swap area.

PROGRAM
bblist - badblock list

USAGE
bblist

FUNCTION
'bblist' will prompt the user for the type of drive and then ask for the cylinder and head combinations which can be obtained from the 'media defect list' provided with the drive. 'bblist' will list badblock numbers for the entire track specified by the cylinder and head combinations given.

'bblist' will output to standard out, but also may be used in conjunction with the '-f' option on mkfs by re-directing its output to the file '/.BBLIST'. (see mkfs (maint)).

SEE ALSO
badblk (maint), mkfs (maint)

BUGS

PROGRAM
clri - clear an index record on a device

USAGE
clri <device> <indexl> ...

FUNCTION
For each index listed in its argument list, 'clri' will fill that index record with zeroes. This should only be done on file systems that are not mounted. The system should be rebooted if done on the root file system.

FILES

DIAGNOSTICS
The command will have exit status of zero if all went well, -1 otherwise.

SEE ALSO
icheck (maint)
dcheck (maint)
fsck (maint)

BUGS

PROGRAM

dcheck - directory consistency check

USAGE

dcheck [-i numbers] [filesystem]

FUNCTION

'dcheck' looks in each directory and checks the link count of each file, including directories, against the number of times that file is referenced, and reports any discrepancies. If the counts differ, 'dcheck' lists the index number, the link count, and the number of references to the file. 'dcheck' also reports if files have been allocated, and there is a zero link count with no references.

If there are index numbers following the -i option, 'dcheck' will list the given number, the index number of the directory it appears in, and the name of the entry.

If a filesystem is not given, the default filesystem will be used instead.

DIAGNOSTICS

If the link count is less than the number of references, a potential problem exists when the link count goes to zero. The remaining references will point to a block with garbage in it. In that case the extra references should go away.

If the link count is greater than the number of references, no damage will occur, but some blocks that have been allocated may never be reclaimed.

FILES

/dev/rootfs; default filesystem

SEE ALSO

icheck(maint), clri(maint), ncheck(maint), filesys(files)

BUGS

Extraneous diagnostics may be given if 'dcheck' is run on an active filesystem.
Needs an 8K stack.

PROGRAM

disktest - direct access device diagnostic

USAGE

disktest special-file

FUNCTION

Disktest allows any direct access device (and some sequential devices) to be tested while UNIX is still available to other users. Care should be taken to make the device which is to be tested unavailable to other system users. The single required argument is the name of the device which is to be tested (ie. /dev/rk1).

Disktest recognizes the following single character commands some of which may be followed by numeric arguments which are assumed to be octal if they begin with a zero and decimal otherwise.

Command Argument Default Meaning

l 1 0 Low block number to test

h 1 1024 High block number to test

n 0 1 Number of blocks per transfer

p 1 or 2 0 Pattern flag

0 default patterns
1 block number is pattern
2 next argument is pattern

o 1 1 Output flag

0 output on errors only
1 extended output

d 0 - Display test parameters

t 1 1 Test repetition factor

g 0 - Execute write, read, compare test

c 0 - Execute a read, compare test

r 0 - Execute a read test

w 0 - Execute a write test

q 0 - Quit - return to the shell

s 1 0 Sequential access

0 sequential test
1 random test

An interrupt (rubout) signal to disktest causes it to print the current test number, block number, and pattern number. A quit signal causes disktest to abort the current test and await a new command. Bits transferred per error are reported on rubout and at the end of each test.

PROGRAM
dump - file system backup

USAGE
dump [-ui0fslL] [dumpfile] [size] [label] [filesystem]

FUNCTION
'dump' is used to create backup copies of a filesystem. It is possible to dump either the entire filesystem or to dump only those files that have changed since the last dump was made.

The actions of 'dump' are controlled by the following arguments:

u If 'dump' is successful (no errors occurred), write the date of the dump and the name of the filesystem on the dump record file. This date is used to determine the starting date when doing incremental dumps.

i Make an incremental dump of the filesystem. All files that have changed since the dump date stored in the dump record file are dumped.

0 (zero) Dump all files from the filesystem.

f Use the next argument as the device or file to write the dump onto.

s The size (in blocks) of the media that the dump is being placed on is taken from the next argument. This is used to determine the number of reels that will be needed for the dump. If the size is negative, the first write error will end the reel and this size then becomes the size of future reels. The user is prompted to change reels as needed. Without this flag, write errors will terminate the dump.

l The next argument is taken to be an identifying label that is recorded in the dump header block of each reel. A '#' character is required in the label to mark where the reel number is to be placed.

L The user is to be prompted for a label. The label must still contain the '#' character.

The default action of 'dump' is:

dump -ilf "REGULUS dump reel ##" /dev/flp0 /dev/rootfs

Various information about the dump is printed before the actual writing of the dump. The user is asked for acknowledgment before 'dump' actually begins to write to

the dump device. If the user does not respond with a 'y', the dump will terminate.

Bad blocks on the filesystem

A 'badblock' file may be specified by setting the ownerid of the file to (-1). 'dump' will dump the index records for this file but will not attempt to read the data blocks, which presumably are blocks which may not be read from the device.

Care must be taken if a dump of a filesystem with a 'badblock' file is restored to a different device than it was originally dumped from, as the 'badblock' file index record is re-created exactly as it was before. If the blocks are not bad on the new filesystem, duplicate entries will occur. These may be fixed by removing the incorrect 'badblock' file and running 'fsck (maint)'.

FILES

/dev/rootfs default filesystem
/dev/flp0 default dumpfile device
/etc/dtab dump record file

DIAGNOSTICS

Errors messages are meant to be self-explanatory and usually are cause for terminating the dump. The filesystem should be quiescent and without 'icheck (maint)' or 'dcheck (maint)' errors.

If the filesystem is in active use, phase errors may occur. A phase error occurs when a file that has been marked for dumping is changed before it is actually dumped. This is not usually a fatal error.

SEE ALSO

restor (maint)
icheck (maint)
dcheck (maint)
dump (files)

BUGS

Phase errors (files being removed between the time that they are marked for dumping and actually dumped) can cause failures.

PROGRAM

fixidx - print and change index records

USAGE

fixidx [device] num ...

FUNCTION

'fixidx' is used to display index record information in a readable format and then allow the user to step through each of the data items and change them as needed. If the first argument does not begin with a numeral, it is assumed to be the name of a block special file which contains a REGULUS filesystem. If no device is specified, '/dev/rootfs' is assumed. All remaining arguments are treated as index record numbers and the following information is displayed:

flags	(octal)
size	(long decimal)
ownerid	(decimal)
groupid	(decimal)
number of links	(decimal)
data block numbers	(decimal)

After the index record data is displayed, the user is stepped through each item and prompted for a change. A carriage return means no change. A decimal number is taken to be the new value for that field. At the end of the fields, the new index record information is displayed and the user is prompted for a write. If he responds with a 'y', the new index record is written. Any other response causes 'fixidx' to forget the changes and continue.

FILES

/dev/rootfs default filesystem device

DIAGNOSTICS

Self-explanatory

SEE ALSO

fsdb (maint)
prtidx (maint)

BUGS

Should be executable only as the superuser.
Using this on a mounted filesystem can be touchy if not in a single-user mode.

PROGRAM
fsck - file system consistency check

USAGE
fsck [-rsynv] [-b <block numbers>] [filesystem ...]

FUNCTION
"fsck" checks the consistency of a Regulus file system. The consistency checks performed are: every block is allocated or on the free list, no block is allocated more than once, every block number is valid within the file system, every index block has a valid index magic number, an appropriate file type and the correct number of links. Basically this coalesces the functions of icheck and dcheck.

If any problem with the file system is found, the user will be queried for whether to fix the problem or not. Typically the fix is to delete the offending index block from the file system (i.e. remove all directory references to the index).

The typical output of 'fsck' is:
The number of missing blocks (not in free list or a file).
The number of special files.
The total number of files on the file system.
The total number of large files.
The total number of directories.
The total number of named pipes.
The number of index records.
Total number of blocks used.
Total number free blocks.

The following options are available:

v Verbose mode, display additional information about the file system including the number of sparse files, the block number of the highest used block, etc.

s Rebuild the free list by putting all unallocated blocks on the free list, and rewriting the super block of the file system with the new free list. The file system should not be mounted at this time. If that is not possible, as in the case of the root file system, the file system should not have any activity on it, and REGULUS should be booted without doing a "sync" so that the in core copy of the super block that is maintained by REGULUS is not written out to the file system. The messages listed above will not be printed if the -s option is given.

b Print out a diagnostic whenever the block numbers specified appear in a file.

y A YES answer, 'y', is assumed for every user query.

n A NO answer, 'n', is assumed for every user query.

r An attempt to recover any lost files and directories will be made. First, all unallocated blocks are checked for being a directory index record. If any is found, the ".." chain is followed until either an allocated directory is encountered or an error occurs. If an allocated directory is found, the highest level unallocated directory is linked into the allocated directory with the unallocated directory's index record number for its name. If an error occurs, the highest level directory will be linked into the "/lost+found" directory with its index record number as its name. Then, the unallocated blocks are scanned for any lost files. Any found lost files will be linked into the "/lost+found" directory with its index record number as its name. Lost special files, named pipes and "unlinked" files (ones with no valid block pointers in the first 61K bytes) are not recovered. Finally the free list is automatically rebuilt. Although an attempt is made to adjust all "." and ".." entries and link counts, multiple passes may be required to get a totally fixed file system. Files and directories with duplicate blocks are recovered as is, so care should be taken when dealing with them. The directory "/lost+found" is automatically created and linked into "/" if it does not exist, and it is automatically extended as needed. If any blocks are needed in this process or in extending found parent directories, they are allocated from the top of the file system. Obviously, if these blocks were allocated to a recovered file, they will be duplicates and should be treated with extreme caution. Every block on the unallocated list is read once in both passes, so this is a very time consuming option.

If no file system argument is given fsck will use the default file system "/dev/rootfs".

FILES

/dev/rootfs is the default file system.

DIAGNOSTICS

Each block that is found to be a duplicate of a previously allocated block is reported, and a sum of all duplicate blocks is also reported. The -b option of 'fsck' can then be used to determine the index blocks that use the blocks listed. Then 'ncheck' can be used to find the actual path names of those index blocks.

A "bad" block is one that is out of range of the file system.

All other diagnostic messages are meant to be self explanatory.

SEE ALSO
dcheck(maint), ncheck(maint), file system(files)

NOTES
Needs an 8K stack.

PROGRAM
fscp - file system copy

USAGE
fscp from to

FUNCTION
'fscp' is used to copy a file system from one device to another.

SEE ALSO
setfssize(cmnd)

PROGRAM
fsdb - file system debugger

USAGE
fsdb filesystem

FUNCTION
'fsdb' is the REGULUS interactive file system debugger. It allows you to look at the blocks of the 'filesystem' with two different display modes, and several different display formats.

The two display modes are:

file	treats the blocks like a file which it steps through starting at the first block number of the index record of the file.
block	assumes that you are simply stepping through the blocks of the file system itself, and not a file.

When in 'file' display mode, you are prompted by a question mark to determine if you wish to continue to display the next block of the file. A 'y' or carriage return tells 'fsdb' to display the next block. A 'q' tells 'fsdb' to stop looking at this file. You are still in the file display mode however.

In both block display mode and file display mode, the current display format will remain the same as the last format unless explicitly changed.

There are several different display formats available:

dir	display block formatted like a directory.
index	display block formatted like an index record.
free	display block as if it were an indirect block of the free list
text	display block as if it contained text
super	display as if it were a file system super block
dump	display block formatted as a dump header.
hex	display the block in hex words
ascii	display the block as ascii bytes
decimal	display the block as decimal words
octal	display the block as octal words

The prompt from 'fsdb' is the block number of the current block followed by a '>'. 'fsdb' commands consist of an optional block number, followed by an optional command. The commands are any one of the two display modes 'file' and 'block', the display formats, or the following:

inc	increment the link count of a directory

	block
dec	decrement the link count of a directory block
clear	in dir mode it prompts for block number, then clears the entry in the block whose block number you entered.
type	tries to determine what kind of block it is.
flist	display the indirect block numbers of the free list

FILES

DIAGNOSTICS

SEE ALSO
filesys(files), directory(files)

BUGS

PROGRAM
getty - set typewriter mode during login

USAGE
/etc/getty type

FUNCTION
'getty' is used by the initialization program to set up the typewriter mode for a terminal after it is opened. The type argument is a character that informs 'getty' of the characteristics of the terminal as specified in the file '/etc/ttys'.

The letters and characteristics are as follows:

0 a CIT-80 or equivalent terminal working at 9600 baud.
1 a CIT-101 or equivalent terminal working at 9600 baud.
2 a 300 baud dialin line.
3 a 1200 baud dialin line.
4 a 1200-300 baud dialin line (try 1200 baud first).
5 a 300-1200 baud dialin line (try 300 baud first).

If anything other than these, the default will be a 9600 baud CIT-80.

After the line is set up, 'getty' will print the contents of the file '/etc/sitename' if this file exists. This is used to provide users logging on an idea of what machine they are using if multiple machines are present. After the 'sitename' information, 'getty' prompts for a user name. A null (break) or end-of-file on cycling lines will cause the next speed to be tried. If the user name ends in a newline, 'getty' will pass that name to 'login' to complete the login process.

FILES
/etc/sitename site information (512 bytes maximum)

DIAGNOSTICS
'getty' does not report errors, but simply does what it is told blindly.

SEE ALSO
init (maint)
ttys (files)
login (cmnd)

BUGS

PROGRAM

icheck - file system consistency check

USAGE

icheck [-s] [-b <block numbers>] [filesystem]

FUNCTION

"Icheck" will check the storage consistency of a filesystem by building a bit map of all the blocks in the file system that appear in files. Icheck then looks at the free list, and compares this with the bit map. Any inconsistencies are reported. Normal output from icheck is as follows:

The number of missing blocks (not in free list or a file).
The number of special files.
The total number of files on the filesystem.
The total number of large files.
The total number of directories.
The number of index records.
Total number of blocks used.
Total number free blocks.

If the -s option is specified, "Icheck" will rebuild the free list by putting all those blocks not already in files on the free list, and rewriting the super block of the file system with the new free list. The file system should not be mounted at this time. If that is not possible, as in the case of the root file system, the file system should not have any activity on it, and REGULUS should be booted without doing a "sync" so that the in core copy of the super block that is maintained by REGULUS is not written out to the filesystem. The messages listed above will not be printed if the -s option is given.

If the -b option is specified on the command line, "Icheck" will print out a diagnostic whenever the block numbers specified appear in a file.

If no filesystem argument is given icheck will use a default filesystem.

FILES

/dev/rootfs is the default filesystem.

DIAGNOSTICS

Each block that is found to be a duplicate of a previously allocated block is reported, and a sum of all duplicate blocks is also reported. The -b option of 'icheck' can then be used to determine the names of all files that use the blocks listed.

A "bad" block is one that is out of range of the filesystem.

All other diagnostic messages are meant to be self explanatory.

SEE ALSO
dcheck(maint), ncheck(maint), filesystem(files)

BUGS
Needs an 8K stack.

PROGRAM
init - process initialization program

USAGE
/etc/init

FUNCTION
'init' is the first program to be invoked by REGULUS after startup. It is always process id 1. The job of 'init' is to start a process for each terminal that is to be used.

The 'init' sequence is in 3 phases. The first phase involves the startup file '/etc/rc'. If this file is found, 'init' will execute a shell with this file as input. This file normally contains command lines to perform housekeeping tasks such as mounting filesystems and starting daemons. The deamon '/etc/update' should be started as a background task at this point.

Should any of the commands in the rc file exit with a non-zero status, the shell will immediately terminate and 'init' will print a message on the console and attempt to return to a single use mode by executing the program '/etc/initl'. Failures in the rc file are usually due to file system problems and are considered serious. Once 'init' falls back to single user mode, the system must be re-booted.

After the rc file is complete, 'init' now opens the file '/etc/ttys' for a description of the terminal lines to use.

For each terminal that is marked ON in the '/etc/ttys' file (refer to 'ttys (files)' document), 'init' forks and attempts to open the terminal. This open will normally be delayed until the terminal is turned on or carrier is established (for a dialup line). When the open succeeds 'init' overlays itself with the program '/etc/getty' to read the users name and continue with the login process.

When the shell or other program that was running on the terminal exits, either by the user logging out or other error (such as a hangup) the original 'init' (process 1) is notified. This process will remove any entry for that terminal from the list of current users in '/etc/utmp' and then repeat the whole sequence again.

'init' catches the HANGUP signal (see 'signal (sys)') and interprets it to mean that the terminal description file '/etc/ttys' should be re-examined. The command sequence for this is:

```
kill -1 1
```

If an entry has been changed from ON or DRAIN to OFF, the process running for that terminal is killed. If an entry

has been changed from ON to DRAIN or the type of terminal for an ON tty has changed, 'init' sends the process a particular user signal. This is caught by 'login' and 'getty' and causes them to exit. If an entry has been changed from OFF to ON, a new process is started for that terminal. Entries that are unchanged are ignored.

It is therefore possible to bring up new terminals (or take down terminals) without re-booting the system simply by changing the entry in '/etc/ttys' and sending the hangup signal to 'init'.

To change the speed of a given line, you must first turn the line OFF, notify 'init' (via the HANGUP signal), then change the speed character and turn the line to ON, again notifying 'init'.

FILES

/etc/rc	init startup file
/etc/ttys	REGULUS terminal configuration file
/etc/utmp	list of current users
/etc/getty	program to invoke when terminal awakens
/etc/initl	program to use if rcfile fails
/initrg	REGULUS startup file (linked to init or initl)

DIAGNOSTICS

Self-explanatory.

SEE ALSO

initl (maint) - single user initialization program
ttys (files)
getty (maint)

BUGS

PROGRAM

initl - single user initialization program

USAGE

/etc/initl

FUNCTION

'initl' is used to invoke REGULUS in a single user mode.

This mode is most useful for fixing the filesystem, which must be done without the update daemon '/etc/update (maint)'.

Single user mode is arranged by linking (see 'ln (cmnd)') the program '/etc/initl' to '/initrg' (the REGULUS startup program). When REGULUS is next booted, 'initl' will immediately execute a shell as the super-user. No startup file processing is done, and in particular, the disk update deamon '/etc/update' is NOT started.

Single user mode may also be reached if the startup file of 'init' reports an error status.

'initl' searches a fixed list of names for a shell to run. The current list is as follows:

```
/bin/sh
/bin/nsh
/usr/bin/sh
/usr/bin/nsh
/sh
```

When a shell starts, 'initl' will wait for it to terminate. When this happens, 'initl' will search a list of multi-user programs for a program to run. If none of these programs can be executed, an error message is printed and the list of shells is searched again.

The current list of multi-user programs is:

```
/etc/init    < no message >            NOWAIT
/bin/login   "REGULUS single user"     WAIT
/bin/sh      "REGULUS single root"     NOWAIT
/bin/login   "REGULUS Maintence Mode"  WAIT
```

'initl' will print the message provided with the name of each of these programs before it attempts to execute it. If the 'wait' flag is 'NOWAIT', 'initl' will overlay itself with the task that it executes. If this task ever exits, NO user programs will be running and the system must be booted.

If the flag is 'WAIT', 'initl' will wait for the task to exit, then start it up again. This provides a minimal single user execution sequence.

In the "Maintenence" mode, the program 'login' will allow

only the users 'root' and 'maint' to log in to the system. This mode is for file system repair.

FILES

/initrg	the REGULUS startup program
/etc/init	multi-user initialization program
/dev/console	

DIAGNOSTICS

Self-explanatory

SEE ALSO

init (maint)

NOTES

The link to '/initrg' must be done manually by the user. If it is not re-linked to the multi-user initialization program, REGULUS will boot up single user the next time also.

If the device "/dev/console" does not exist, 'initl' may be unable to either start or notify the user of the problem. A failure of REGULUS to provide a program prompt when it has successfully loaded from the disk may often be traced to a missing console device.

PROGRAM
lpd - line printer daemon

USAGE
lpd

FUNCTION
The line printer daemon is responsible for the actual printing of files that have been spooled by 'lpr (cmnd)'.

FILES
/usr/spool/lpr/dfaXXXXX
contains name of spooled file to print and is created by lpr. The XXXXX is a unique number formed from the pid of the lpr process then created it.

/usr/spool/lpr/lock
prevents more than one daemon from running at the same time.

DIAGNOSTICS
The diagnostics produced by 'lpd' are intended to be self explanatory.

SEE ALSO
lpr (cmnd)

BUGS
If the system crashes, 'lpd' lock files are often left behind, thus preventing new daemons from starting up again. It is advisable to remove these locks by placing appropriate commands in the system start-up file '/etc/rc'.

PROGRAM
mkfs - make a REGULUS file system

USAGE
mkfs special size [badblocks || -f]

FUNCTION
'mkfs' creates a REGULUS file system on the file 'special', and gives it 'size' number of blocks. 'mkfs' creates the super block, and builds the free list. 'mkfs' creates the first index block of the root directory, the root directory itself, and creates entries for `.' and `..' in it.

If any badblock numbers are given to 'mkfs', these blocks are left out of the free list when it is written on the filesystem. If the number of badblocks exceeds that which is allowed as an input argument, the '-f' option may be used. With '-f' mkfs will read the ascii file '/.BBLIST' to obtain the badblock values. The file '/.BBLIST' is a list of badblocks separated by spaces or tabs. (See also bblist (maint)).

FILES
filesys(files), directory(files).

DIAGNOSTICS

SEE ALSO
filesystem (files), badblk (maint), bblist (maint)

BUGS

PROGRAM

mknod - make a special file node

USAGE

mknod name [p|[[b|c] major minor]]

FUNCTION

'mknod' creates a special file node or named pipe. Special file nodes are used to reference devices. Named pipes are the same as pipelines obtained from the 'pipe (sys)' system call, except they have a valid index record and file system name associated with them, and do not go away on the last close of the pipeline.

If the type specified is 'b', 'mknod' will create a block type special file. If the type specified is 'c', 'mknod' will create a character type special file. If the type specified is 'p', a named pipe file will be created.

Block and character special files may only be created by the super-user. Named pipes may be created by anyone.

FILES

DIAGNOSTICS

SEE ALSO

filesystem (files)
pipe (sys)

BUGS

PROGRAM

mount - mount a filesystem on a directory

USAGE

mount [-r] [all] || { [special file] [file] }

FUNCTION

The 'mount' command is used to notify REGULUS that a mountable file system is present on some directory. 'mount' takes 0 to 3 arguments:

none with no arguments, 'mount' will simply format and print the contents of the mounted file table, /etc/mtab.

-r mount filesystems as read-only. This flag must be specified if a drive is to be physically write protected to prevent REGULUS from attempting to write on the device.

If there is a single argument, it must be one of the following:

all the file '/etc/mountfiles' is examined and all entries in that file with the 'a' (all) flag specified are then mounted. 'mount' will print a message for each filesystem mounted.

file the file '/etc/mountfiles' is searched for a line with either a matching special file name or directory name. If a match of either is found, that filesystem will be mounted per the flags found in /etc/mountfiles or specified via -r.

If there are 2 arguments (other than -r), they are assumed to be a directory name and a special file name. They may appear in either order. 'mount' will attempt to mount the given special file on that directory. If the special file name is not a full path name (i.e., does not start with the char '/') 'mount' will prepend the string '/dev/' to the name before any other actions. If the device is not in /dev, full path names are required to prevent this.

FILES

/etc/mountfiles	mounted filesystem description file
/etc/mtab	currently mounted filesystem table

DIAGNOSTICS

Various system errors may occur. 'mount' will print an explanation for any failures to mount the filesystem.

SEE ALSO

umount (maint)
mountfiles (files)
mtab (files)

BUGS

Must be set-uid-root.

PROGRAM

mvdir - move a directory

USAGE

/etc/mvdir olddirname newdirname

FUNCTION

'mvdir' is used to change the name of directories. 'mvdir' is called with the current directory name followed by the new name you wish the directory to be named. It is legal to move directories anywhere within a filesystem.

The name of the new directory may not exist before executing 'mvdir'. It is not allowed for either directory names to be subsets of the other (for example, /usr/tests may not be moved to /usr/tests/cmnd).

Only the super-user may use mvdir.

To move a directory within its current parent directory, all users may use 'mv(cmnd)'.

SEE ALSO

mv(cmnd)
mkdir(cmnd)

BUGS

PROGRAM

ncheck - print file name and index number

USAGE

ncheck [-a] [-i num ...] [device]

FUNCTION

'ncheck' prints the names and corresponding index record number for files on a filesystem. If no device is specified, '/dev/rootfs' is used.

The '-i' flag introduces a list of index record numbers. If this flag is not used, all file names and their index record numbers are printed. This flag must be followed by at least one number. The names of only the files corresponding to these index record numbers are then printed.

The '-a' flag causes index record numbers to be printed for '.' and '..' entries. These are normally suppressed.

FILES

/dev/rootfs default file system

DIAGNOSTICS

SEE ALSO

icheck (maint)
dcheck (maint)
dump (cmnd)
restor (cmnd)

BUGS

Needs an 8K stack.

PROGRAM

prtidx - print index records

USAGE

prtidx [device] num ...

FUNCTION

'prtidx' is used to display index record information in a readable format. If the first argument does not begin with a numeral, it is assumed to be the name of a block special file which contains a REGULUS filesystem. If no device is specified, '/dev/rootfs' is assumed. All remaining arguments are treated as index record numbers and the following information is displayed:

flags	(octal)
size	(long decimal)
ownerid	(decimal)
groupid	(decimal)
number of links	(decimal)
data block numbers	(decimal)

FILES

/dev/rootfs default filesystem device

DIAGNOSTICS

Self-explanatory

SEE ALSO

fsdb (maint)
fixidx (maint)

BUGS

PROGRAM
rclock - use the real-time clock

USAGE
rclock [-s] [-r]

FUNCTION
'rclock' is used to either set the system time from the real-time clock, or to set the real-time clock from the current REGULUS system time. By default, with no arguments, 'rclock' will read the real-time clock and set the system time from it. The two possible flags for 'rclock' are:

-s Set the system time from the real-time clock. This is the default.

-r Set the real-time clock from the REGULUS system time.

SEE ALSO
date(cmnd)
time(sys)
readrclock(subs)
setrclock(subs)

DIAGNOSTICS
You need to be super-user to set the REGULUS system time.

BUGS

PROGRAM
restor - restore a filesystem from a dump

USAGE
restor [-uwfrx] [dumpfile] [filesystem] [index # ...]

FUNCTION
'restor' is used to rebuild a filesystem from a previously made filesystem dump (see 'dump (maint)'). It is also used in conjunction with 'dump' to change the size of a filesystem.

The following arguments are understood:

f The dump is read from the device or file which is the next argument.

r Use the next argument as the name of the filesystem to restore the dump onto. This filesystem must be a formatted REGULUS filesystem. See 'mkfs (maint)' for more on making a filesystem.

x Interpret the following arguments as index record numbers and extract the files with these index record numbers. The files will be named with their respective index record numbers.

w Wait for an acknowledgement from the user before reading from the dump device. If possible, 'restor' will shut down any motors to facilitate the changing of media in the dump device.

u Update the dump table simulating a full dump. This is used when the full dumps are being done in a different manner than with 'dump (maint)', such as stand-alone tape or disk cartridge.

The default action of 'restor' is:

restor -fr /dev/flp0 /dev/rootfs

When restoring a filesystem, you should first restore a complete filesystem dump. Then apply each following incremental dump until the filesystem is restored to the appropriate date. It is important that the incremental restores are done in sequence to insure that the filesystem is back to the original state.

To change the size of a filesystem, first make a complete dump of the filesystem. Then do a 'mkfs (maint)' on the filesystem with the new filesystem size. And finally 'restor' the filesystem from the dump just made.

Bad blocks on the filesystem

If the device that is being restored to has known bad blocks, these blocks should be identified before the restore is attempted. Write errors caused by bad blocks will cause 'restor' to fail. See 'mkfs (maint)' and 'badblk (maint)' for descriptions on removing bad blocks from the filesystem.

If the device that is being restored to is the same device where the dump was originally made, any 'badblock' files will be re-created with the same bad block numbers contained in them.

FILES

/dev/rootfs default filesystem name
/dev/flp0 default dumpfile name

DIAGNOSTICS

Information from the dumpfile header is printed before the dump begins. Other error messages are meant to be self-explanatory and usually result in termination of the dump. The only exception is when changing reels in a multi-reel dump. The reel number is compared to the expected reel number until the right reel is found.

SEE ALSO

dump (maint)
mkfs (maint)
dump (files)

NOTES

The -x flag is not yet implemented.

BUGS

Most errors require that you start from the beginning again.

PROGRAM

setfssize - set filesystem size in superblock

USAGE

setfssize <device> <size>

FUNCTION

The file system size in the super block of the file system on the <device> is changed to <size>. Only the superuser can execute this command. This should be done carefully and should be followed by an "fsck -s <device>" command to rebuild the free list.

PROGRAM

shutdown - shut down user processing

USAGE

shutdown [-rsk] [-m"message"] [-t<seconds>] [-<count>]

FUNCTION

'shutdown' gracefully shuts down processing. shutdown repeatedly sends a message to every logged on user every so many seconds until either every user is logged off or the specified count is expired. If the -k option is specified, then a Terminate signal is sent to every task. If the -r option is specified, an automatic reboot is attempted. If the -s option is specified, the system will exit to the debugger. The default number of seconds is 30, the default number of tries is 3, and the default message is "System shutting down, please log off".

SEE ALSO

wall (cmnd), stop (cmnd)
boot (sys)

BUGS

PROGRAM
stop - stop REGULUS and exit to hardware debugger

USAGE
stop

FUNCTION
'stop' unconditionally stops Regulus and exits to the debugger. It waits for all scheduled block I/O to complete before exiting. 'stop' does not do an automatic sync, and is considerably less graceful than 'shutdown'.

SEE ALSO
wall (cmnd), shutdown (cmnd)
boot (sys)

BUGS

PROGRAM
sync - update system "dirty" buffers

USAGE
sync

FUNCTION
This call causes all information currently residing in memory that should be stored on disk to be written out.

SEE ALSO
update(CMND), init(CMND)

BUGS

PROGRAM
umount - unmount a mounted filesystem

USAGE
unmount [all]

unmount [-]filename

FUNCTION
The 'unmount' command is used to notify REGULUS that a mounted file system is no longer to be used. 'unmount' has 1 argument:

all 'unmount' will examine the mounted file table '/etc/mtab' and unmount each entry. This may or may not be a complete list of the mounted filesystems since it is possible for the table to become corrupted. In any case, 'unmount' will print a line for each filesystem that is unmounted.

filename 'unmount' will examine the mounted file system table '/etc/mtab' for an entry with a corresponding name. If one is found, the file system is unmounted.

If the filename is not a full path name (ie., does not start with '/') 'unmount' will prepend the string '/dev/' before any actions. If this is not desired, full path names or the '-' flag must be used.

To handle exceptional cases, if the filename is preceeded by a dash '-', the file is assumed to be a special file and it is unmounted even if no entry is found in the mount table.

FILES
/etc/mtab currently mounted filesystem table

DIAGNOSTICS
Various system errors may occur. 'unmount' will print an explanation for any failures to unmount the filesystem.

SEE ALSO
mount (maint)
mtab (files)

NOTES
Must be set-uid-root.

PROGRAM

update - disk I/O integrity task

USAGE

/etc/update

FUNCTION

'update' syncs the disk every 30 seconds. This insures that in the event of a crash, the disk is at most 30 seconds out of date.

'update' is normally run out of the '/etc/rc' startup file at the time that REGULUS is brought up in a multi-user mode.

FILES

DIAGNOSTICS

SEE ALSO

init (maint)
initl (maint)
sync (maint)
/etc/rc (files)

BUGS

PROGRAM
wall - write to all users

USAGE
/etc/wall

FUNCTION
'wall' first reads from its standard input until it receives an end-of-file. Then, it sends this information to every user that is logged in preceded by "Broadcast message from ...".

The user of this program should be the super-user in order to override any protections users may have against receiving messages.

FILES
/etc/utmp - list of currently logged in users
/dev/tty* - terminal special files

SEE ALSO
mesg(cmnd), write(cmnd), shutdown(cmnd)

DIAGNOSTICS
"Can't open /dev/tty?" when trying to open a user's tty file fails.

BUGS

MISCELLANEOUS

This section contains a summary of the commands available in the REGULUS line editor program, the format of floating point numbers with other miscellaneous information.

NAME

config - configuring REGULUS run-time parameters

FUNCTION

REGULUS contains a configuration table which is used each time the system is started. This configuration specifies such things as how many disk buffers are to be allocated, how many task slots are allocated in the task table, etc. The entries in the configuration table may be changed and such a change will affect both the REGULUS system size and performance. In general, increasing the values in the configuration table will make REGULUS use more memory and make it run faster or support more users and tasks. The following is a list of the variables in the configuration table and their meanings:

```
long fswapblock;  //first block in swap area
int nindex;       //max number of unique open files
int nfiles;       //max total number of open files
int nsignals;     //max number of pending signals
int ntasks;       //max number of active tasks
int nbufs;        //number of disk cache buffers
int ncbufs;       //number of character queue buffers
int nfree;        //number of active file systems
int mapsize;      //max main memory size in K bytes
int maxqsize;     //maximum character output queue size
int max2bufs;     //numb of tasks allowed 2 disk buffers
int nlocks;       //max number of active file locks
int ndelays;      //max number of device delays
int maxcputime;   //user clock ticks per lowering priority
int maxsystime    //system clock ticks per lowering priority
int nlebufs;      //number of line edit buffer headers
int nsegs;        //number of shared or sticky segments
int vaddmax;      //max user virtual address in K
int maxcomm;      //max size of user common area in K
int commuid;      //common match on UID flag
int coretime;     //seconds to hold task in memory
int swaptime;     //seconds to hold task on swap
int ncomms;       //number of shared data common segments
long minsbrk;     //minimum break size in bytes
int maketbsize;   //max size of maketask args in bytes
long stkincr;     //increment to add to stack on overflow
```

Remember that in REGULUS C integers are 16 bits in length and longs are 32 bits in length.

The values for these parameters can be determined and modified using ddt. The command sequence:

```
ddt /regulus
nbufs/d
;f
```

will display the current value of nbufs in decimal. The command sequence:

```
ddt /regulus
35!nbufs
;f
```

will change the value of nbufs to 35 decimal. The parameters which affect the size and performance of the system most dramatically are: nbufs (disk cache buffers -- 542 bytes each), ntasks (number of active tasks -- 50 bytes each), nindex (number of open files -- 26 bytes each), nfiles (number of successful file opens -- 48 bytes each), and ncbufs (number of character queue buffers -- 16 bytes each).

SEE ALSO

ddt (cmnd)

NAME

as - assembler for the Motorola 68000

FUNCTION

The REGULUS assembler 'as' is designed to be used in conjunction with the other REGULUS standard language processing tools: cc and lo. The language implemented is primarily as is described in any of the standard 68000 assembler manuals, with a few enhancements to make it accept some of the more standard assembler mnemonics. It will attempt to optimize instructions to take advantage of certain 68000 instructions (for example the add quick instruction).

The following instructions are recoginized to aid the assembly language programmer by making the assembly language more regular: clr.x An (where x is 1,b, or w) is allowed and will actually generate a suba.x An,An instruction; add, sub, cmp with an A register source/destination are allowed and generate adda, suba, cmpa; add, and, cmp, eor, or, sub are allowed with immediate first operands and actually generate addi, andi, cmpi, eori, ori, subi instructions if the second operand is not register direct; any shift instruction with no shift count specified assumes a shift count of one (ie. "asl r1" is equivalent to "asl #1,r1"); the mnemonics 'inc' and 'dec' are also recognized and are functionally equivilent to "addq 1" or "subq 1". Several mnemonics have been added to the standard set of condition code instructions: 'bt' maps to 'bra', 'bhs' maps to 'bhis', 'bnz' maps to 'bne', 'bze' maps to 'beq', 'dbhs' maps to 'dbhi', 'dblo' maps to 'dbcs', 'dbnz' maps to 'dbne', 'dbze' maps to 'dbeq', 'shs' maps to 'scc', 'slo' maps to 'scc', 'snz' maps to 'sne', and 'sze' maps to 'seq'.

Several optimizations take place. All branch instructions generate short relative branches where possible, including forward references. 'jsr' instructions are changed to 'bsr' instructions if the resulting 'bsr' is shorter than the 'jsr'. 'move', 'add', 'sub' mnemonics will actually generate 'moveq', 'addq', and 'suba' instructions where possible. If a 'move' instruction rather than a 'moveq' instruction is desired (affecting only lower byte or word of D register), the size attribute must be explicitly coded ie. move.b or move.w. The assembler will change any move or move.l to moveq if possible.

All labels must terminate with a ':', unless they start in column 1. Registers may be referenced as r0 - r15, R0 - R15, D0 - D7, d0 - d7, A0 - A7, or a0-a7. Registers R8 - R15 are the same as A0 - A7. Numbers are interpreted as decimal. Hexidecimal may be specified by preceding the number by a dollar sign ($) and numbers which begin with a preceeding zero (0) are assumed to be octal.

REGULUS assembler directives can be specified in upper and/or lower case characters and may be preceded by an optional dot (.). The following directives have been implemented:

text
data
bss
: These three directives instruct the assembler to change the assembler base segment to the text, data, or bss segment respectively. Each assembly starts in the text segment. It is illegal to assemble instructions or data into the bss segment. Symbols may be defined and storage may be reserved using the .ds directive in the bss segment.

section #
: This directive is used to define a base segment like the .bss, .data and .text directives discussed above. The sections can be numbered 0 to 15 inclusive. Section 14 maps to data, section 15 is bss and all the others are text sections.

globl name[,name...]
xdef name[,name...]
xref name[,name...]
: These directives make the name(s) external. If they are defined in the current assembly, this statement makes these names available to other routines during a load by lo68. If these names are not defined in the current assembly, they become undefined external references and lo68 will link them to external values of the same name in other routines. Specifying the -u flag will force the assembler to make all undefined names external.

comm name,expression
: This directive defines a named block common area. When several routines are linked together with lo68, all block common areas with the same name are loaded at the same address. The size of this block common storage area is the largest value of the expression part of all 'comm' directives with the same name. No error message is produced if the areas are of different sizes.

even
: If the location counter is odd, it is incremented by one so that the next instruction or data field will begin on an even memory boundary.

The relocation counter may be manipulated with a statement like:

*=expr
Care should be exercised when using this facility. The expression may only move the relocation counter forward. The unused space is filled with zeroes in the text or data segments and is simply not assigned in the bss segment. This facility requires the assembler to not allow comment lines which begin with "*=". Comments beginning with "* =" are allowed.

org expression
This directive is used to change the program counter to the specified address. The expression must not contain any external references and must be computable at the time of assembly.

ds[.x] operand
This is the directive used to define storage. Storage can be defined in bytes, words or long words depending upon the extension specified ('b', 'w', 'l' where word is the default). The memory is not initialized, just reserved. Word or long defined storage will be aligned on a word boundary.

comline expression
This directive is identical to the 'ds.b' instruction. The specified argument must be a constant expression.

equ expression
set expression
Currently 'set' and 'equ' are implemented exactly the same. These directives are used to assign a value to the specified label. This non-optional label may not be redefined elsewhere in the program. It can then be used anywhere that a constant is required and will be evaluated to the specified value.

end
The 'end' directive specifies the end of the assembler text. It is not required for appropriate assembly. If it is not at the end of the text an error will be reported.

offset expression
This directive creates a dummy storage section using the 'ds' directive. No code generating instructions may occur. It is terminated by a 'section', 'data', 'bss', 'text' or 'end'. The offset table begins at the address specified in the expression. The 'set', 'equ', 'reg', 'xref', 'xdef', 'globl', 'offset', and any conditional directive may appear inside.

dcb[.x] size,value
Just like the dc.b command. Defines a constant block of the specified size of bytes, words or longs.

reg reglist
This directive builds a register mask which can be used by the movem instruction. One or more registers can be listed in increasing order of the form : 'R?[-R?[/R?[-R?...]...]]'. Where R? can be replaced by A0-A7, D0-D7 and R0-R15. The register list might look like : A2-A4/A7/D1/D3-D5. The registers may also be designated separated by commas (eg. A1,A2,D5,D7).

Conditional assembly directives can have any level of nesting. The following conditional assembler directives have been implemented:

ifeq expression
ifne expression
ifle expression
iflt expression
ifge expression
ifgt expression
The expression is tested against zero (with ifeq: equal, ifne: not equal, ifle: less or equal, iflt: less than, ifge: greater or equal, ifgt: greater) and if it is evaluated true then the code enclosed is assembled, otherwise the code is ignored until the matching endc is found.

ifc 'string1','string2'
ifnc 'string1','string2'
The two strings are compared. The 'c' condition is true if they are exactly the same, the 'nc' condition is true if they do not match.

endc
Signifies the end of the code to be conditionally assembled.

ASCII string constants may be enclosed in single quotes (ie. 'ABCD') or in double quotes (ie. "ac14").

Several directives are recognized, but ignored: 'mask2', 'idnt', 'ttl', 'opt', and 'page'.

SEE ALSO
as(cmnd)

NAME

environment - user environment

DESCRIPTION

The user environment is an array of strings passed by 'maketask (sys)'. These strings have the format "name=value" where name is the environment variable name and value is the current value. The initial user environment is set up by 'login (cmnd)' which sets "PATH=/bin:/usr/bin" and HOME to the login directory. The 'sh (cmnd)' program can then alter its environment and the environment of its children with its 'setenv' command. The current environment can be displayed with the 'env (cmnd)' program, and the env program can also execute tasks with a specified environment other than the current one. A user program may access its environment by either an additional argument block pointer in 'main' or by the 'getenv (subs)' user subroutine.

Some commonly used environment variables are:

PATH The sequences of directory prefixes for searching for executable programs separated by colons (not prefixed). This is used by 'sh (cmnd)', 'time (cmnd)', 'env (cmnd)', etc. The default is: "PATH=/bin:/usr/bin" set by 'login (cmnd)'.

HOME The user's home directory, set by 'login (cmnd)'

TERM The terminal type.

SEE ALSO

env (cmnd)
sh (cmnd)
maketask (sys)
getenv (subs)

NAME

ledit - REGULUS line editor summary

LINE EDIT SUMMARY

Control Character	Definition
a	Backspace new line one character.
b	Replace old line with new line; do not terminate edit.
c	Copy one character from old line to new line.
d	EOF. Insert EOF character and terminate edit.
e	Toggle insert mode. Begin insert echos "<"; end ">".
f	Copy the remaining characters from the old line to the new line echoing each and then terminate the edit.
g	BELL.
h	Backspace new line one character.
i	Tab. A tab stop is defined every four characters. Same as TAB key on many terminals.
j	Line feed. Terminate the edit.
k	Backspace new line (only) through all characters until a letter or digit is encountered, then through all letters or digits until a special character is encountered. Backspaces new line one "symbol Echo "\" for each unit backspaced.
l	Copy remaining characters of old line to new line with echoing; do not terminate the edit.
m	Carriage return. Terminate the edit.
n	Backspace old line and new line one character.
o	Copy characters from old line to new line up the first occurrence of the next character typed. "CTRLob" copies the line to the next "b" in the old line.
p	Delete (skip) characters from the old line to the next occurrence of the the next character typed. "CTRLpf" deletes all characters on the old line up to the next "f". Echos "%" for each character deleted.
q	XON. Enable terminal output.
r	Display the remainder of the old line and all of the new line.

s XOFF. Disable terminal output.

t Delete (skip) one character from the old line.
Echos "%" for the deleted character.

u Delete the new line and reset old line to the beginning.
Echos "\" followed by a carriage return.

v Escape character. Turns off any special meaning which may be associated with the following character. Thus, to transmit a " enter "CTRLvCTRLa"; the echoed character will be "&A".

w Backspace new line (only) through all blanks to the first nonbl character; then, backspace through all nonblanks to the first Backspace a "word".
Echos " \" for each unit backspaced.

x Delete (skip) characters from the old line through the first occurrence of the next character typed. "CTRLxs" deletes through the next "s".

y Terminate the edit, releasing the new line without updating the old line.

z Copy characters from the old line to the new line through the next occurrence of the next character typed.
"CTRLz)" copies through the next ")".

NOTES:

The old line normally contains the last line which the user typed. If the "e" command is used in the text editor (see ed(I)), the old line is the last line edited.

The control character is entered by pressing the CTRL key and then simultaneously depressing the desired function key, e.g., CTRLa.

To disable the line editor, see stty(I), -ledit option.

NAME

users - adding new users to a REGULUS system

FUNCTION

When adding a new user, you must
1) make an entry in /etc/passwd.
2) make that user a home directory using 'mkdir (cmnd)'.
3) change the owner of the home directory to the new user using 'chown (cmnd)'.

To make an entry in /etc/passwd, edit the file /etc/passwd and copy the format of the user lines already there. Each line in this file identifies user information and consists of six fields separated by colons ":".

Field	Use
1	user's login name.
2	user's password. This should be left blank at first since the password is stored in this field in an encrypted form. When logging in as a user who has a null password, login will ask for a password -- the proper response is just carriage return.
3	user id number which must be unique.
4	group id number which need not be unique.
5	name of the user's home directory.
6	name of the shell for this user. If this field is blank, 'login (cmnd)' will use '/bin/sh'.

REGULUS is delivered with a null password for the superuser "root". This should probably be changed quickly as none of the protection mechanisms are enforced for the superuser.

To have an additional terminal enabled in multi-user, you must edit the /etc/ttys file, see 'ttys (files)'

To add an addtional group, you must edit the /etc/group file, see 'group (files)'.

SEE ALSO

ed (cmnd)
mkdir (cmnd)
chown (cmnd)
passwd (files)
ttys (files)
group (files)

NAME

pindex - making a permuted index

DESCRIPTION

A permuted index, such as the one at the beginning of the REGULUS manual, can be made using the REGULUS commands kwic, sort, and unrot. This document will describe how we would create a permuted index for the subroutine (subs) section of this manual.

All REGULUS subroutine documentation can be found in /doc/man/subs. First we need to take the NAME line out of each file, using grep(cmnd):

```
grep " - " * > names
```

Which will create the file "names" which will look something like this:

```
alloc:alloc - C storage allocator
amatch:amatch - regular expression pattern matcher
```

Next, we must remove the filename, at the beginning of each line with the ed command

```
1,$s/?*://
```

Then the edit command

```
1,$s/ - /(subs)&/
```

should be given to insert "(subs)" after the command name. The file should then look like this:

```
alloc(subs) - C storage allocator
amatch(subs) - regular expression pattern matcher
```

After this is done, the following command should be typed:

```
kwic < names | sort -d | unrot > INDEX
```

The result of this command will create a file 'INDEX' which should contain the permuted index.

SEE ALSO

kwic(cmnd)
sort(cmnd)
unrot(cmnd)

NAME

/dev/asN - Atasi Winchester Disk interface

FUNCTION

The name /dev/as20 refers to the Atasi 33Mb or 46 Mb winchester disk drives supported on the Alcyon APS system.

The Atasi disk drives have five or seven fixed platters. Each platter has 256 bytes/sector, 32 sectors/track and 645 tracks. This means each platter is 5.16Mb or 10,320 512 byte blocks.

One may partition the atasi, or any disk sub-system. There is a table in the bootable REGULUS program "/regulus" called "dskparam" which contains disk partion information. This table may be modified using ddt, the same as for modifying configuration parameters. The dskparam table is 16 entries long, each entry consists of a long (32-bit) entry for the starting block offset, and a long (32-bit) entry for the number of blocks in this partition. The lower four bits of the minor device number is used as an index into the partition table. The next most significant two bits are the OMTI controller logical unit number, and the next most significant two bits are an index into the drive type: 0 is atasi, 1 is syquest, 2 is dma. Thus minor device 17 is logical unit 1, typically the removable disk, partition 1.

The standard entries for the dskparam table are:

0	0	72240	/dev/as20 - atasi disk drives
1	0	9792	/dev/syNN - syquist disk drives
2	0	10560	/dev/dma[rf] - dma disk drives
3	0	0	not used
4	0	0	not used
5	0	0	not used
6	0	0	not used
7	0	0	not used
8	0	0	not used
9	0	0	not used
10	0	0	not used
11	0	0	not used
12	0	0	not used
13	0	0	not used
14	0	0	not used
15	0	0	not used

Although the standard dskparam table reflects partitioning the disks for their actual physical layout, there is no restriction on this. For example, an atasi disk may be configured to look line one 46Mb disk drive, or two 23 Mb partitions. There is even no restriction about partitions overlapping, although this is not recommeded. However, the standard configuration will probably be adequate for all

but the most unusual needs.

Block I/O is accomplished by using the above special files. All I/O to the floppy is turned into requests for logical sectors. All requests to the block device are turned into 512 byte reads or writes no matter how many characters the "read" or "write" system call requests. The starting sector address for any I/O is always on a REGULUS blocksize boundary. The blocksize is 512 bytes.

Raw I/O is accomplished by using the name /dev/ras20. This I/O is only allowed in 512 byte blocks on 512 byte block boundaries. This bypasses the REGULUS buffering mechanism and DMA's the data directly to or from the user's memory.

NAME

/dev/dma[rf] - DMA Winchester Disk interface

FUNCTION

The names /dev/dmar and /dev/dmaf refer to the DMA disk drives supported on the Alcyon APS system. on the Alcyon APS system.

The DMA disk drive has one fixed platter and one removable platter. Each platter has 256 bytes/sector, 33 sectors/track and 640 tracks. This means each platter is 5.28Mb or 10,560 512 byte blocks.

One may partition the DMA, or any disk sub-system. There is a table in the bootable REGULUS program "/regulus" called "dskparam" which contains disk partion information. This table may be modified using ddt, the same as for modifying configuration parameters. The dskparam table is 16 entries long, each entry consists of a long (32-bit) entry for the starting block offset, and a long (32-bit) entry for the number of blocks in this partition. The lower four bits of the minor device number is used as an index into the partition table. The next two most significant bits are the OMTI controller logical unit number, and the next most significant bits are the disk type: 0 is atasi, 1 is syquist and 2 is dma.

The standard entries for the dskparam table are:

0	0	72240	/dev/as20 - atasi disk drives
1	0	9792	/dev/syNN - syquist disk drives
2	0	10560	/dev/dma[rf] - dma disk drives
3	0	0	not used
4	0	0	not used
5	0	0	not used
6	0	0	not used
7	0	0	not used
8	0	0	not used
9	0	0	not used
10	0	0	not used
11	0	0	not used
12	0	0	not used
13	0	0	not used
14	0	0	not used
15	0	0	not used

Although the standard dskparam table reflects partitioning the disks for their actual physical layout, there is no restriction on this. For example, an DMA disk may be configured to look line one 5Mb disk drive, or two 2.5 Mb partitions. There is even no restriction about partitions overlapping, although this is not recommeded. However, the standard configuration will probably be adequate for all but the most unusual needs.

Block I/O is accomplished by using the above special files. All I/O to the floppy is turned into requests for logical sectors. All requests to the block device are turned into 512 byte reads or writes no matter how many characters the "read" or "write" system call requests. The starting sector address for any I/O is always on a REGULUS blocksize boundary. The blocksize is 512 bytes.

Raw I/O is accomplished by using the names /dev/rdmar and /dev/rdmaf. This I/O is only allowed in 512 byte blocks on 512 byte block boundaries. This bypasses the REGULUS buffering mechanism and DMA's the data directly to or from the user's memory.

NAME

/dev/tty[0-3] - Signetics SC2681 DUART Serial interface

FUNCTION

The names /dev/tty0, ..., /dev/tty3 refer to the Signetics SC2681 DUART serial ports on the Alcyon A68KPM Processor Board. /dev/console is another name for /dev/tty0.

Baud rates are software programmable, however, only the following are supported:

50
110
135
200
300
600
1200
2400
4800
9600

In addition, character size (5, 6, 7 and 8 bits), parity and one or two stop bits are fully programmable. This is also the default setting upon system boot.

SEE ALSO

tty (dev)
stty (cmnd)

NAME

/dev/lp - line printer interface

FUNCTION

The name /dev/lp refers to the line printer interface on the Motorola Debug Module board. This is a Centronics compatable interface.

The user program 'stty (cmnd)' may change the settings of the line printer interface. Line delays are ignored, and hardware setting programming is ignored. It is typically better to access the line printer though the spooler program 'lpr (cmnd)' rather than to access the device explicitly.

SEE ALSO

stty (cmnd)
lpr (cmnd)

NAME

/dev/mem - access to physical memory

FUNCTION

The name /dev/mem is an interface to all of the systems physical memory. The /dev/mem name can be opened, read, written and seeked just like any disk file. This is how the user command 'ps (cmnd)' can access other processes memory. No checks are made of valid addresses, hence /dev/mem can access I/O memory, with potentially disastrous results.

It is not recommeded that anyone but the superuser have write access to /dev/mem.

/dev/mem most typically is accessed as 16 bit words, although reads or writes of one byte will be guaranteed to to a MC68000 byte move.

NAME

/dev/null - null device

FUNCTION

The name /dev/null is an interface to the "null" device. The /dev/null name can be opened, read, written and seeked just like Any read from /dev/null results in an end of file, 0 bytes read, condition. Any write to /dev/null will return the number of bytes requested to write.

/dev/null is most typically used as a "byte bucket" for programs whose output is not needed.

NAME

/dev/tty - general terminal interface

FUNCTION

The file /dev/tty refers to the controlling terminal of a process group. It may be used by programs to insure that their output may directed to the terminal no matter how output has been redirected and also by programs that need a specific file name.

All terminals use the same general interface. This interface is controlled by the 'stty (cmnd)' user program and the 'ioctl (sys)' system call. The form of the ioctl call which affect a terminals controlling parameters is:

```
#include <sys/ioctl.h>
#include <termio.h>
ioctl(fd,cmd,arg)
int fd;                 file descriptor for tty
int cmd;                command, one of the following
struct termio *arg;     pointer to user termio buffer
```

The commands available to ioctl using this form are:

TCGETA — Get the terminal parameters

TCSETA — Set the terminal parameters

TCSETAW — Set the terminal parameters after waiting for output to drain

TCSETAF — Wait for output to drain, flush the input queue then set the parameters.

The structure of the terminal controlling buffer, which is defined in <termio.h> is:

```
#define NCC 8
struct termio {
    int c_iflag;        /* input modes */
    int c_oflag;        /* output modes */
    int c_cflag;        /* hardware control modes */
    int c_lflag;        /* line discipline settings */
    char c_line;        /* line discipline */
    char c_cc[NCC];     /* control characters */
};
```

The following modes are those for the standard Unix line discipline. Some System III features are not implemented in REGULUS, they are indicated by: (NA) Those features that are not supported in hardware are typically ignored, rather than

causing an error.

The c_iflag entry defines the terminal input modes, which are:

IGNBRK	0000001	Ignore break condition (NA)
BRKINT	0000002	Signal interrupt on break (NA)
IGNPAR	0000004	Ignore characters with parity errors
PARMRK	0000010	Mark characters with DEL, NUL, x
INPCK	0000020	Input parity check
ISTRIP	0000040	Strip bit 8 of 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	Any character restarts output
IXOFF	0010000	Enable start/stop input control

The c_oflag entry defines the terminal output modes, which are:

OPOST	0000001	Postprocess output
OLCUC	0000002	Map lower case to upper case on output
ONLCR	0000004	Map NL to CR/NL on output
OCRNL	0000010	Map CR to NL on output
ONOCR	0000020	No CR at column 0
ONLRET	0000040	NL performs CR function
OFILL	0000100	Use fill characters for delay (NA)
OFDEL	0000200	Fill is DEL, otherwise NUL (NA)
NLDLY	0000400	New line delay
NL0	0000000	No NL delay
NL1	0000400	Delay approximately .10 sec.
CRDLY	0003000	Carriage return delay
CR0	0000000	No CR delay
CR1	0001000	Delay approximately .10 sec.
CR2	0002000	Delay approximately .20 sec.
CR3	0003000	Delay approximately .30 sec.
TABDLY	0014000	Tab delay
TAB0	0000000	No tab delay
TAB1	0004000	Delay approximately .10 sec.
TAB2	0010000	Delay approximately .20 sec.
TAB3	0014000	Expand tabs to spaces
BSDLY	0020000	Backspace delay
BS0	0000000	No backspace delay
BS1	0020000	Delay approximately .10 sec.
VTDLY	0040000	Vertical tab delay
VT0	0000000	No vertical tab delay
VT1	0040000	Delay approximately .10 sec.
FFDLY	0100000	Form feed delay
FF0	0000000	No form feed delay
FF1	0100000	Delay approximately .10 sec.

The c_cflag defines the hardware settings for the terminal:

CBAUD	0000017	Baud rate setting
B0	0000000	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
EXTA	0000016	External A (typically 19200 baud)
EXTB	0000017	External B (typically 38400 baud)
CSIZE	0000060	Character size
CS5	0000000	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Two or one stop bits
CREAD	0000200	Enable reciever
PARENB	0000400	Parity enable
PARODD	0001000	Odd or even parity
HUPCL	0002000	Hang up on last close
CLOCAL	0004000	Local or dial-up line
EXOPEN	0010000	Exclusive open of device

The c_lflag defines the settings for the Regulus line editor:

ISIG	0000001	Enable signals
ICANON	0000002	Line editor enable/disable
XCASE	0000004	Allow escaped upper case input (NA)
ECHO	0000010	Enable echo
ECHOE	0000020	Echo erase as BS-SP-BS
ECHOK	0000040	Echo kill as erase characters
ECHONL	0000100	Echo NL (NA)
NOFLSH	0000200	No flush after INTR or QUIT
LEDITO	0000400	Place output chars in line edit old line
TABS	0077000	Programmable tab settings (1-63)
NOLEDIT	0100000	Line editor off, but do erase and kill process

The c_line selects the line discipline. Ultimately, line discipline 0 will be the standard Unix line discipline, line discipline 1 will be the REGULUS line editor discipline. Currently, this entry is ignored and only the REGULUS line editor line discipline is available.

A Unix-like line discipline may be simulated by setting the NOLEDIT bit. This disables most of the REGULUS line editor functions, but continues to do character erase and line kill processing.

The c_cc table defines the special control characters. The positions in the c_cc table are:

VINTR	0	Interrupt character
VQUIT	1	Quit character
VERASE	2	Erase character
VKILL	3	Kill character
VEOF	4	End of file character
VEOL	5	End of line character
VMIN	4	Raw mode number of chars to transfer (NA)
VTIME	5	Raw mode timeout (NA)

Note that most control characters are used by the REGULUS line editor, so that changing any of these to some character that is translated by the line editor will cause that function to be lost from the line editor.

Another form of the ioctl system call is:

```
#include <sys/ioctl.h>
#include <termio.h>
ioctl(fd,cmd,arg)
int fd;                 file descriptor for tty
int cmd;                command, one of the following
int arg;                (optional) argument to cmd
```

The commands available to ioctl using this form are:

TCSBRK — Wait for output to drain, the send a break character (NA).

TCXONC — Xon/xoff control. Suspend output if arg is 0, otherwise restart output if suspended.

TCFLSH — Flush input/output queues. Flush input queue if arg is zero, flush output queue if arg is one, flush both input and output if arg is two.

TCQSIZE — Inquire current queue size. Returns number of characters currently in the input or output queue. If the arg is zero, the input queue size is returned, if the arg is one the output queue size is returned.

TCQAVAIL — Inquire current number of characters

available in queue. Returns the number of characters that can be placed in the input or output queue without losing characters on input or going to sleep on output. If the arg is zero, the number of characters available in the input queue is returned, if the arg is one the number of characters availabe in the output queue is returned.

SEE ALSO
ioctl (sys)