NCR

Programmer Reference Manual

D1-0399-C
October, 1986

16-Bit Release 3.01
32-Bit Release 1.03

It is the policy of NCR Corporation to improve products as new technology, components, software, and firmware become available. NCR Corporation, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult your NCR representative or NCR office for information that is applicable and current.

To maintain the quality of our publications, we need your comments on the accuracy, clarity, organization, and value of this book.

Address correspondence to:

Technical Education/Publications
NCR Corporation
3325 Platt Springs Road
West Columbia, SC 29169
U.S.A.

TABLE OF CONTENTS

2. System Calls

3. **Subroutines**

4. File Formats

5. Miscellaneous Facilities

OVERVIEW

RELEASE: The information in this *Programmer Reference Manual* applies to Release 3.01 of the 16-bit and Release 1.01 of the 32-bit operating systems based on UNIX.

AUDIENCE: The audiences for this book are programmers, analysts, and system support personnel. It is expected that the user is familiar with UNIX, another operating system derived from UNIX, or appropriate operating system courses.

CONTENT: This manual describes the system calls, subroutines, file formats, and miscellaneous facilities of the operating system. For a description of the operating system commands and games, refer to the *User Reference Manual.* For a description of the maintenance commands, special files, and maintenance procedures, refer to the *Superuser Reference Manual.*

The manual is divided into four sections, some containing sub-classes:

2. System Calls
3. Subroutines:
   - 3C. C and Assembler Library Routines
   - 3F. FORTRAN Library Routines
   - 3M. Mathematical Library Routines
   - 3S. Standard I/O Library Routines
   - 3X. Miscellaneous Routines
4. File Formats
5. Miscellaneous Facilities

Section 2 (*System Calls*) describes the entries into the UNIX system kernel, including the C language interface.

Section 3 (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (*File Formats*) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys.**

Section 5 (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several system calls, subroutines, etc. In such cases, the entry appears only once, alphabetized under its major name.

All entries are based on a common format, not all of whose parts always appear:

The NAME part gives the name(s) of the entry and briefly states its purpose.

The SYNOPSIS part summarizes the use of the entry being described. A few conventions are used:

> **Boldface** strings are literals and are to be entered just as they appear.
>
> *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).
>
> Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as name or file, it always refers to a *file* name.
>
> Ellipses ... are used to show that the previous argument prototype may be repeated.

The DESCRIPTION part discusses the subject at hand.

The EXAMPLE(S) part gives example(s) of usage, where appropriate.

The FILES part gives the file names that are built into the program.

The SEE ALSO part gives pointers to related information. References of the form **name(N)**, where N is a number 2 through 5 possibly followed by a letter, refer to entries in this manual. References of the form **name(N)**, where N is the number 1 or 6 possibly followed by a letter, refer to entries in the *User Reference Manual*. References of the form **name(1M)**, **name(7)**, or **name(8)** refer to entries in the *Superuser Reference Manual.*

The DIAGNOSTICS part discusses the diagnostic indications that may be produced. Messages that are self-explanatory are not listed.

The WARNINGS part points out potential problems.

The RESTRICTIONS part gives known restrictions and deficiencies. Occasionally, the suggested fix is also described.

The SUPPORT STATUS part specifies the item as supported or not supported. The operating system is fully supported. Included in the system, however, are unsupported items you might find useful. For example, SVS-FORTRAN, DI-3000 Graphics, and a line printer spooler are supported for the system. The UNIX FORTRAN, graphics, and spooler items are included as unsupported items.

A table of contents, a permuted index derived from that table, and a module contents are included in this manual.

The *permuted index* is a combined index for the *User Reference Manual*, *Programmer Reference Manual*, and *Superuser Reference Manual.* On each *permuted index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

A *module contents*, defining the components of each operating system module, is also included. The operating system is distributed as several modules any or all of which may be installed on your system. Before using this manual, check with your system administrator to determine which modules are installed on your system. The *module contents* is a combined list for entries in the *User Reference Manual*, *Programmer Reference Manual*, and *Superuser Reference Manual*.

All entries are available on-line via the *man*(1) command, if these files are installed during software installation.

PERMUTED INDEX

MODULE CONTENTS

The operating system is distributed as a base module and several functionally independent modules. Each independent module provides a specific capability. This *module contents* defines the content of each of the modules of the operating system. Following a short description of each module, a combined contents for entries in the *User Reference Manual*, *Programmer Reference Manual*, and *Superuser Reference Manual* specifies the operating system module which contains the feature (na specifies not applicable). Before using this manual, check with your system administrator to determine which operating system modules are installed on your system.

MODULE DESCRIPTIONS

Business Base Module (bbase)
The *Business Base Module (bbase)* is the minimum operating system. This module contains the basic operating system utilities, menu programs for the system administrator (sa login) and end users, and reconfiguration and maintenance features for the support technician (ncrm login).

Extension Module (exten)
The *Extension Module (exten)* contains non-essential operating system utilities such as *vi*, *spell*, and *csh* that may be useful to some users.

Miscellaneous Module (misc)
The *Miscellaneous Module (misc)* contains further non-essential operating system utilities such as games and the UNIX line printer spooler facility.

Graphics Module (graph)
The *Graphics Module (graph)* contains all UNIX graphics facilities as unsupported software.

System Documentation Module (man)
The *System Documentation Module (man)* includes the online manual pages, related utilities, and the system administrator help files.

Software Development Module (devel)
The *Software Development Module (devel)* includes the facilities for software development such as C language tools, *f77*, and the Source Code Control System (SCCS) facility.

68010 Compiler Module (compile)
The *68010 Compiler Module (compile)* is available only for 32-bit systems and includes the 16-bit C compiler and its associated libraries and routines. On 16-bit systems, this software is in the *devel* module.

System Accounting Module (acct)
The *System Accounting Module (acct)* provides the complete system accounting facility: methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees

to specific logins.

Communication Module (comm)
The *Communication Module (comm)* includes the standard system-to-system teletype communication facilities *uucp* and *cu* and the utilities required to configure and maintain the facilities.

System Encryption Module (crypt)
The *System Encryption Module (crypt)* is available only in the United states and contains the encryption program.

USER REFERENCE MANUAL CONTENTS

1. Commands and Application Programs

300	handle special functions of DASI 300 and 300s terminals	man
4014	paginator for the TEKTRONIX 4014 terminal	man
450	handle special functions of the DASI 450 terminal	man
acctcom	search and print process accounting file(s)	acct
adb	absolute debugger	devel
admin	create and administer SCCS files	devel
ar	archive and library maintainer for portable archives	bbase
as	common assembler	bbase
asa	interpret ASA carriage control characters	exten
ascvt	release 2.x to 3.0 assembler source translator	devel
at	execute commands at a later time	bbase
awk	pattern scanning and processing language	bbase
backup	backup, restore-backup or restore selected files	bbase
banner	make posters	misc
basename	deliver portions of path names	bbase
bc	arbitrary-precision arithmetic language	exten
bdiff	big diff	devel
bfs	big file scanner	misc
bs	a compiler/interpreter for modest-sized programs	misc
cal	print calendar	exten
calendar	reminder service	bbase
cancel	cancel requests to an LP line printer	misc
cat	concatenate and print files	bbase
cb	C program beautifier	devel
cc	C compiler	bbase
cc.10	C compiler	compile
cd	change working directory	bbase
cdc	change the delta commentary of an SCCS delta	devel
cflow	generate C flow graph	devel
chmod	change mode	bbase
chown	change owner or group	bbase
clear	clear terminal screen	bbase
cmp	compare two files	bbase
col	filter reverse line-feeds	bbase
comb	combine SCCS deltas	devel
comm	select or reject lines common to two sorted files	exten

cp	copy, link or move files	bbase
cpio	copy file archives in and out	bbase
cpp	the C language preprocessor	bbase
crontab	user crontab file	bbase
crypt	encode/decode	crypt
csh	a shell (command interpreter) with C-like syntax	exten
csplit	context split	devel
ct	spawn getty to a remote terminal	bbase
ctags	create a tags file	devel
ctrace	C program debugger	devel
cu	call another UNIX system	bbase
cut	cut out selected fields of each line of a file	bbase
cw	prepare constant-width text for troff	man
cxref	generate C program cross-reference	devel
date	print and set the date	bbase
dc	desk calculator	exten
dd	convert and copy a file	bbase
delta	make a delta (change) to an SCCS file	devel
deroff	remove nroff/troff, tbl, and eqn constructs	misc
diff	differential file comparator	bbase
diff3	3-way differential file comparison	exten
diffmk	mark differences between files	misc
dircmp	directory comparison	bbase
dis	disassembler	devel
du	summarize disk usage	bbase
dump	dump selected parts of an object file	devel
echo	echo arguments	bbase
ed	text editor	bbase
edit	text editor (variant of ex for casual users)	exten
enable	enable/disable LP printers	misc
env	set environment for command execution	bbase
eqn	format mathematical text for nroff or troff	man
error	analyze and disperse compiler error messages	exten
ex	text editor	exten
expr	evaluate arguments as an expression	bbase
f77	Fortran 77 compiler	devel
factor	factor a number	exten
file	determine file type	bbase
find	find files	bbase
fsplit	split f77, ratfor, or efl files	devel
gdev	graphical device routines and filters	graph
ged	graphical editor	graph
get	get a version of an SCCS file	devel
getopt	parse command options	bbase
graph	draw a graph	graph
graphics	access graphical and numerical commands	graph
greek	select terminal filter	man
grep	search a file for a pattern	bbase
gutil	graphical utilities	graph
head	give first few lines	exten
help	ask for help	devel

hp	handle special functions of HP 2640 and 2621-series terminals	graph
id	print user and group IDs and names	bbase
ipcrm	remove a message queue, semaphore set or shared memory id	bbase
ipcs	report inter-process communication facilities status	bbase
join	relational database operator	exten
kill	terminate a process	bbase
last	indicate last logins of users and teletypes	bbase
ld	link editor for common object files	bbase
lex	generate programs for simple lexical tasks	devel
line	read one line	bbase
lint	a C program checker	devel
login	sign on	bbase
logname	get login name	bbase
look	find lines in a sorted list	exten
lorder	find ordering relation for an object library	devel
lp	send requests to an LP line printer	misc
lpstat	print LP status information	misc
ls	list contents of directory	bbase
ls	list contents of directory (Berkeley) •	bbase
m4	macro processor	devel
machid	provide truth value about your processor type	bbase
mail	send mail to users or read mail	bbase
mailx	interactive message processing system	bbase
make	maintain, update, and regenerate groups of programs	bbase
makekey	generate encryption key	exten
man	print entries in this manual	man
mesg	permit or deny messages	bbase
mkdir	make a directory	bbase
mklost+found	make a lost+found directory for fsck	bbase
mkstr	create an error message file by massaging C source	devel
mm	print/check documents formatted with the MM macros	man
mmt	typeset documents, viewgraphs, and slides	man
more	file perusal filter for crt viewing	bbase
newform	change the format of a text file	exten
newgrp	log in to a new group	bbase
news	print news items	bbase
nice	run a command at low priority	bbase
nl	line numbering filter	exten
nm	print name list of common object file	bbase
nohup	run a command immune to hangups and quits	bbase
nroff	format or typeset text	man
od	octal dump	bbase
pack	compress files	exten
packsf	compress and uncompress sparse file	exten
passwd	change login password	bbase
paste	merge same lines of several files or subsequent lines of one file	exten
pg	file perusal filter for soft-copy terminals	bbase
pr	print files	bbase
print	line printer spooler	bbase

prof	display profile data	devel
prs	print an SCCS file	devel
ps	report process status	bbase
ptx	permuted index	man
pwd	working directory name	bbase
ratfor	rational Fortran dialect	devel
regcmp	regular expression compile	devel
rev	reverse lines of a file	bbase
rm	remove files or directories	bbase
rmdel	remove a delta from an SCCS file	devel
sact	print current SCCS file editing activity	devel
sag	system activity graph	acct
sar	system activity reporter	acct
sccsdiff	compare two versions of an SCCS file	devel
sdb	symbolic debugger	devel
sdiff	side-by-side difference program	exten
sed	stream editor	bbase
sh	shell, the standard/restricted command programming language	bbase
shl	shell layer manager	exten
size	print section sizes of common object files	devel
sleep	suspend execution for an interval	bbase
sln	link files symbolically	bbase
sno	SNOBOL interpreter	misc
sort	sort and/or merge files	bbase
spell	find spelling errors	exten
spline	interpolate smooth curve	graph
split	split a file into pieces	bbase
spool	spool queue manager	bbase
ssp	make output single spaced	bbase
stat	statistical network useful with graphical commands	graph
strings	find the printable strings in a object, or other binary, file	devel
strip	strip symbol and line number information from object file	devel
stty	set the options for a terminal	bbase
su	become superuser or another user	bbase
sum	print checksum and block count of a file	bbase
sync	update the super block	bbase
tabs	set tabs on a terminal	misc
tail	deliver the last part of a file	bbase
tar	tape file archiver	bbase
tbl	format tables for nroff or troff	man
tc	phototypesetter simulator	misc
tee	copy input to standard output and to files	bbase
test	condition evaluation command	bbase
time	time a command	bbase
timex	time a command; report process data and system activity	acct
toc	graphical table of contents routines	graph
touch	update access and modification times of a file	devel
tpcvt	filter for old streaming tape format	misc

tplot	graphics filters	graph
tput	query terminfo database	bbase
tr	translate characters	bbase
true	provide truth values	bbase
tsort	topological sort	exten
tty	get the name of the terminal	bbase
ul	underline output for a terminal	bbase
umask	set file-creation mode mask	bbase
uname	print name of current UNIX system	bbase
unget	undo a previous get of an SCCS file	devel
uniq	report repeated lines in a file	exten
units	interactive conversion program	exten
uucp	UNIX system to UNIX system copy	bbase
uustat	uucp status inquiry and job control	bbase
uuto	public UNIX-to-UNIX system file copy	bbase
uux	UNIX-to-UNIX system command execution	bbase
val	validate SCCS file	devel
vc	version control	devel
vi	screen-oriented (visual) display editor based on ex	exten
wait	await completion of process	bbase
wc	word count	bbase
what	identify SCCS files	bbase
whereis	locate source, binary, and or manual for program	man
who	who is on the system	bbase
write	write to another user	bbase
xargs	construct argument list(s) and execute command	bbase
xstr	extract strings from C programs to implement shared strings	devel
yacc	yet another compiler-compiler	devel

6. Games

arithmetic	provide drill in arithmetic problems	misc
back	the game of backgammon	misc
bj	the game of black jack	misc
craps	the game of craps	misc
maze	generate a maze	misc
moo	guessing game	misc
ttt	tic-tac-toe	misc
wump	the game of hunt-the-wumpus	misc

PROGRAMMER REFERENCE MANUAL CONTENTS

2. System Calls

access	determine accessibility of a file	devel
acct	enable or disable process accounting	devel
alarm	set the process alarm clock for a process	devel
brk	change data segment space allocation	devel
chdir	change working directory	devel
chmod	change mode of file	devel
chown	change owner and group of a file	devel
chroot	change root directory	devel

close	close a file descriptor	devel
creat	create a new file or rewrite an existing one	devel
dup	duplicate an open file descriptor	devel
exec	execute a file	devel
exit	terminate process	devel
fcntl	file control	devel
ffp	floating point processor access	devel
fork	create a new process	devel
getpid	get process, process group, and parent process IDs	devel
getuid	get real user, effective user, real group, and effective group IDs	devel
ioctl	control device	devel
kill	send a signal to a process or a group of processes	devel
link	link to a file	devel
lockf	provide exclusive file regions for reading and writing	devel
lseek	move read/write file pointer	devel
mknod	make a directory, or a special or ordinary file	devel
mount	mount a file system	devel
msgctl	message control operations	devel
msgget	get message queue	devel
msgop	message operations	devel
nice	change priority of a process	devel
open	open for reading or writing	devel
pause	suspend process until signal	devel
pipe	create an interprocess channel	devel
plock	lock process, text, or data in memory	devel
profil	execution time profile	devel
ptrace	process trace	devel
pwrnote	power recovery notification	devel
pwrtime	power recovery interval to single-user state	devel
read	read from file	devel
rmnt	mount and unmount directorys across file systems	devel
semctl	semaphore control operations	devel
semget	get a set of semaphores	devel
semop	semaphore operations	devel
sernum	get serial number of current system	devel
setpgrp	set process group ID	devel
setuid	set user and group IDs	devel
shmctl	shared memory control operations	devel
shmget	get shared memory segment	devel
shmop	shared memory operations	devel
signal	specify what to do upon receipt of a signal	devel
slink	link files across file systems	devel
stat	get file status	devel
stime	set time	devel
swrite	synchronously write on a file	devel
sync	update super-block	devel
time	get time	devel
times	get process and child process times	devel
tslice	set/get time slice	devel
ulimit	get and set user limits	devel
umask	set and get file creation mask	devel

umount	unmount a file system	devel
uname	get name of current UNIX system	devel
unlink	remove directory entry	devel
ustat	get file system statistics	devel
utime	set file access and modification times	devel
wait	wait for child process to stop or terminate	devel
write	write on a file	devel

3. Subroutines

a64l	convert between long integer and base-64 ASCII string	devel
abort	generate an IOT fault	devel
abort	terminate Fortran program	devel
abs	return integer absolute value	devel
abs	Fortran absolute value	devel
aimag	Fortran imaginary part of complex argument	devel
aint	Fortran integer part intrinsic function	devel
assert	verify program assertion	devel
bessel	Bessel functions	devel
bool	Fortran bitwise boolean functions	devel
bsearch	binary search	devel
clock	report CPU time used	devel
conjg	Fortran complex conjugate intrinsic function	devel
conv	translate characters	devel
crypt	generate DES encryption	devel
ctermid	generate file name for terminal	devel
ctime	convert date and time to string	devel
ctype	classify characters	devel
curses	screen functions with optimal cursor motion	devel
curses	CRT screen handling and optimization package	devel
cuserid	get character login name of the user	devel
dial	establish an out-going terminal line connection	devel
dim	positive difference intrinsic functions	devel
dprod	double precision product intrinsic function	devel
drand48	generate uniformly distributed pseudo-random numbers	devel
ecvt	convert floating-point number to string	devel
end	last locations in program	devel
erf	error function and complementary error function	devel
exp	Fortran exponential, logarithm, square root intrinsic functions	devel
exp	exponential, logarithm, power, square root functions	devel
fclose	close or flush a stream	devel
ferror	stream status inquiries	devel
flcvt	float format conversions	devel
floor	floor, ceiling, remainder, absolute value functions	devel
fopen	open a stream	devel
fread	binary input/output	devel
frexp	manipulate parts of floating-point numbers	devel
fseek	reposition a file pointer in a stream	devel
ftw	walk a file tree	devel
ftype	explicit Fortran type conversion	devel

gamma	log gamma function	devel
getarg	return Fortran command-line argument	devel
getc	get character or word from stream	devel
getcwd	get path-name of current working directory	devel
getenv	return value for environment name	devel
getenv	return Fortran environment variable	devel
getgrent	get group file	devel
getlogin	get login name	devel
getopt	get option letter from argument vector	devel
getpass	read a password	devel
getpw	get name from UID	devel
getpwent	get password	devel
gets	get a string from a stream	devel
getut	access utmp file entry	devel
hsearch	manage hash search tables	devel
hypot	Euclidean distance function	devel
iargc	number of command line arguments	devel
index	return location of Fortran substring	devel
l3tol	convert between 3-byte integers and long integers	devel
ldahread	read the archive header of a member of an archive file	devel
ldclose	close a common object file	devel
ldfhread	read the file header of a common object file	devel
ldgetname	retrieve symbol name for common object file symbol table entry	devel
ldlread	manipulate line number entries of a common object file function	devel
ldlseek	seek to line number entries of a section of a common object file	devel
ldohseek	seek to the optional file header of a common object file	devel
ldopen	open a common object file for reading	devel
ldrseek	seek to relocation entries of a section of a common object file	devel
ldshread	read an indexed/named section header of a common object file	devel
ldsseek	seek to an indexed/named section of a common object file	devel
ldtbindex	compute index of symbol table entry of object file	devel
ldtbread	read an indexed symbol table entry of a common object file	devel
ldtbseek	seek to the symbol table of a common object file	devel
len	return length of Fortran string	devel
lockf	record locking on files	devel
logname	return login name of user	devel
lsearch	linear search and update	devel
malloc	main memory allocator	devel
malloc	fast main memory allocator	devel
matherr	error-handling function	devel
max	Fortran maximum-value functions	devel
mclock	return Fortran time accounting	devel
memory	memory operations	devel
min	Fortran minimum-value functions	devel
mktemp	make a unique file name	devel

mod	Fortran remaindering intrinsic functions	devel
monitor	prepare execution profile	devel
nlist	get entries from name list	devel
perror	system error messages	devel
plot	graphics interface subroutines	graph
popen	initiate pipe to/from a process	devel
printf	print formatted output	devel
putc	put character or word on a stream	devel
putenv	change or add value to environment	devel
putpwent	write password file entry	devel
puts	put a string on a stream	devel
qsort	quicker sort	devel
rand	simple random-number generator	devel
rand	Fortran uniform random-number generator	devel
regcmp	compile and execute regular expression	devel
round	Fortran nearest integer functions	devel
scanf	convert formatted input	devel
setbuf	assign buffering to a stream	devel
setjmp	non-local goto	devel
sign	Fortran transfer-of-sign intrinsic function	devel
signal	specify Fortran action on receipt of a system signal	devel
sleep	suspend execution for interval	devel
sputl	access long numeric data in a machine independent fashion.	devel
ssignal	software signals	devel
stdio	standard buffered input/output package	devel
stdipc	standard interprocess communication package	devel
strcmp	string comparision intrinsic functions	devel
string	string operations	devel
strtod	convert string to double-precision number	devel
strtol	convert string to integer	devel
swab	swap bytes	devel
system	issue a shell command from Fortran	devel
system	issue a shell command	devel
termcap	terminal independent operation routines	devel
tmpfile	create a temporary file	devel
tmpnam	create a name for a temporary file	devel
trig	Fortran trigonometric intrinsic functions	devel
trig	trigonometric functions	devel
trigh	Fortran hyperbolic intrinsic functions	devel
trigh	hyperbolic functions	devel
tsearch	manage binary search trees	devel
ttyname	find name of a terminal	devel
ttyslot	find the slot in the utmp file of the current user	devel
ungetc	push character back into input stream	devel
vprintf	print formatted output of a varargs argument list	devel
vprintf	print formatted output of a varargs argument list	devel

4. File Formats

a.out	common assembler and link editor output	na
acct	per-process accounting file format	na
alert	error records for devices exceeding threshold values	na

alertmesg	logalert summary message file	bbase
ar	common archive file format	na
checklist	list of file systems processed by fsck	na
core	format of core image file	na
cpio	format of cpio archive	na
dir	format of directories	na
errfile	error-log file format	na
filehdr	file header for common object files	na
fs	format of system volume	na
fspec	format specification in text files	na
gettydefs	speed and terminal settings used by getty	bbase
gps	graphical primitive string, format of graphical files	na
group	group file	bbase
inittab	script for the init process	bbase
inode	format of an inode	na
issue	issue identification file	bbase
ldfcn	common object file access routines	na
linenum	line number entries in a common object file	na
master	master device information table	bbase
mnttab	mounted file system table	na
passwd	password file	bbase
plot	graphics interface	na
profile	setting up an environment at login time	na
reloc	relocation information for a common object file	na
rmnttab	mounted directory table	bbase
sccsfile	format of SCCS file	na
scnhdr	section header for a common object file	na
syms	common object file symbol table format	na
sysident	date and release number of the operating system	bbase
term	format of compiled term file	bbase
terminfo	terminal capability data base	bbase
threshold	logalert threshold file	bbase
utmp	utmp and wtmp entry formats	na

5. Miscellaneous Facilities

ascii	map of ASCII character set	man
btermcap	terminal capability data base	misc
environ	user environment	na
eqnchar	special character definitions for eqn and neqn	bbase
fcntl	file control options	devel
font	description files for device-independent troff	man
greek	graphics for the extended TTY-37 type-box	man
man	macros for formatting entries in this manual	man
math	math functions and constants	devel
me	macros for formatting papers	man
mm	the MM macro package for formatting documents	man
mosd	the OSDD adapter macro package for formatting documents	man
mptx	the macro package for formatting a permuted index	man
ms	text formatting macros	man
mv	troff macro package to typeset view graphs and slides	man
prof	profile within a function	devel

regexp	regular expression compile and match routines	devel
stat	data returned by stat system call	na
term	conventional names for terminals	na
termcap	terminal capability data base	bbase
troff	description of output language	na
types	primitive system data types	bbase
values	machine-dependent values	devel
varargs	handle variable argument list	devel

SUPERUSER REFERENCE MANUAL CONTENTS

1. Maintenance Commands and Application Programs

accept	allow/prevent LP requests	misc
acct	overview of accounting and miscellaneous accounting commands	acct
acctcms	command summary from per-process accounting records	acct
acctcon	connect-time accounting	acct
acctmerg	merge or add total accounting files	acct
acctprc	process accounting	acct
acctsh	shell procedures for accounting	acct
arterm	archiver for termcap data bases	bbase
badlist	produce a list of bad blocks for drive	bbase
brc	system initialization shell scripts	bbase
btblock	display contents of bootblock from Winchester disks and	bbase
checkall	faster file system checking procedure	misc
chroot	change root directory for a command	bbase
clri	clear i-node	bbase
cns_filter	console log filter	bbase
config	configure a UNIX system	bbase
cpset	install object files in binary directories	bbase
crash	examine system images	exten
cron	clock daemon	bbase
dcopy	copy file systems for optimal access time	exten
devnm	device name	bbase
df	report number of free disk blocks	bbase
diskusg	generate disk accounting data by user ID	acct
dkpart	set/calculate disk partition sizes	bbase
errdead	extract error records from dump	bbase
errdemon	error-logging daemon	bbase
errpt	process a report of logged errors	bbase
errstop	terminate the error-logging daemon	bbase
ff	list file names and statistics for a file system	bbase
filesave	daily/weekly UNIX system file system backup	bbase
finc	fast incremental backup	bbase
findroot	find root device name	bbase
format	formatter for the 5.25 and 8 inch disks	bbase
formatck	format checker for the 5.25 and 8 inch disks	bbase
frec	recover files from a backup tape	bbase
fsck	file system consistency check and interactive repair	bbase

fsdb	file system debugger	bbase
fuser	identify processes using a file or file structure	exten
fwtmp	manipulate connect accounting records	acct
getty	set terminal type, modes, speed, and line discipline	bbase
icheck	display inode number	bbase
init	process control initialization	bbase
install	install commands	exten
killall	kill all active processes	bbase
ldhpsio	high performance serial	bbase
link	exercise link and unlink system calls	bbase
logalert	error log threshold analysis utility	bbase
lpadmin	configure the LP spooling system	misc
lpd	line printer daemon	bbase
lpsched	start/stop the LP request scheduler and move requests	misc
mkfs	construct a file system	bbase
mknod	build special file	bbase
mount	mount and dismount file system	bbase
mvdir	move a directory	bbase
ncheck	generate names from i-numbers	bbase
newfs	construct a file system	bbase
profiler	operating system profiler	devel
pwck	password/group file checkers	bbase
rmount	mount and unmount directorys across file systems	bbase
runacct	run daily accounting	acct
sadp	disk access profiler	acct
sar	system activity report package	acct
setmnt	establish mount table	bbase
setrmnt	establish rmount table	bbase
shutdown	shutdown - shutdown system	bbase
single	single - go from multi-user to single-user mode	bbase
skyload	load the SKY Floating Point Processor	misc
spooldev	spool system device table manager	bbase
sublock	display contents of superblock from	bbase
tic	terminfo compiler	bbase
uuclean	uucp spool directory clean-up	comm
uusub	monitor uucp network	comm
verchk	display SCCS version number and file attributes	bbase
verify	turn on/off read after write check for device	bbase
volcopy	copy file systems with label checking	bbase
wall	write to all users	bbase
whodo	who is doing what	bbase

7. Special Files

err	error-logging interface	bbase
hpsio	high performance serial	bbase
ios	intelligent 8-channel serial	bbase
lp	line printer	bbase
mem	core memory	bbase
nec	nec —	bbase
null	the null file	bbase
prf	operating system profiler	bbase
sd	driver for the SCSI disks	bbase

ss	driver for the SCSI tapes	bbase
sxt	pseudo-device driver	devel
termio	general terminal interface	bbase
tp	driver for 5.25 and 8 inch streaming tapes	bbase
trace	event-tracing driver	devel
tty	controlling terminal interface	bbase
wd	driver for the 5.25 inch disks	bbase
xl	driver for the 8 inch Winchester disks	bbase

8. Maintenance

boot	startup procedure	na
diag	run in-service diagnostics	bbase
inserv	inservice diagnostics	bbase
l0diag	perform automatic level 0 diagnostics	na
rc	command script for system housekeeping	bbase
sus	startup procedure	na
suscmd	invoke Start-Up-Subsystem function	na
suslog	invoke Start-Up-Subsystem (SUS) log driver	na

NAME

intro — introduction to system calls, definitions, and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

The *SYSTEM CALLS* section of this manual describes all of the system calls. The entry for each system call contains several subsections as outlined below; not all entries contain all subsections.

The *SYNOPSIS* subsection describes the declarations which must be included to use the system call in the C program.

The *DESCRIPTION* subsection describes the function of the system call. Frequently this description includes terminology specific to the use of the system call, such as *effective* and *real user IDs.* The *DEFINITIONS* section of this introduction defines the most frequently used terms in the system calls.

The system call may not always perform the described function. The *FAILURE CONDITIONS* subsection lists reasons why the call may fail, accompanied by the *error name*, such as *NOENT*. Each *error name* has an associated error number. When the system call fails, the correct error number is put in the external variable **errno**. *Successful* calls do not clear **errno** however, so **errno** should be tested only after an *error return*. The *ERRORS* section of this introduction describes errors in detail.

The *RETURN VALUE* subsection lists the values returned by the system calls and their meanings. The *error return*, the value returned on error, is usually −1. *Errno* indicates the specific error which caused the error return.

DEFINITIONS

Process ID

Each active process in the system is uniquely identified by a positive integer called a process ID. An exception to this is the swapper which is actually two processes of the same process ID: the swap in and the swap out processes. The range of the process ID is from 0 to 30,000.

Parent Process ID

A currently active process may create a new process; see *fork*(2). The parent process ID of a process is the process ID of its creator.

Process Group ID

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes; see *kill*(2).

Tty Group ID

Each active process can be a member of a terminal group identified by a positive integer called the tty group ID. This grouping is used to terminate a group of related process upon termination of one of the processes in the group; see *exit*(2) and *signal*(2).

Real User ID and Real Group ID

Each user allowed on the system is identified by a positive integer called a real user ID. Each user is also a member of a group. The group is identified by a positive integer called the real group ID. An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

Effective User ID and Effective Group ID

An active process has an effective user ID and an effective group ID that determine file access permissions (see below). The effective user ID and effective group ID are equal to real user ID and real group ID of the process respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group-ID bit set; see *exec*(2).

Super-user

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

Special Processes

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as *proc0* and *proc1*. *Proc0* is the scheduler. *Proc1* is the initialization process (*init*). Proc1 is the ancestor of every other process in the system and is used to control the process structure.

File Descriptor.

A file descriptor is a small integer used to do I/O on a file. The value of a file descriptor is from 0 to 63. A process may have no more than 64 file descriptors (0-63) opened simultaneously. A file descriptor is returned by system calls such as *open*(2), or *pipe*(2). The file descriptor is used as an argument by calls such as *read*(2), *write*(2), *ioctl*(2), and *close*(2).

File Name.

Names consisting of 1 to 14 characters may be used to name an ordinary file, special file or directory. These characters may be selected from the set of all character values excluding \0 (null) and the ASCII code for / (slash). Note that it is generally unwise to use *, ?, [, or] as part of file names because of the special meaning attached to these characters by the shell. See *sh*(1). Although permitted, it is advisable to avoid the use of unprintable characters in file names.

Path Name and Path Prefix

A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name. More precisely, a path name is a null-terminated character string constructed as follows:

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

where <file-name> is a string of 1 to 14 characters other than the ASCII slash and null, and <dirname> is a string of 1 to 14 characters (other than the ASCII slash and null) that names a directory. If a path name begins with a slash, the path search begins at the *root* directory. Otherwise, the search begins from the current working directory. A slash by itself names the root directory. Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.

Directory.
Directory entries are called links. By convention, a directory contains at least two links, . and .., referred to as *dot* and *dot-dot* respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

Root Directory and Current Working Directory.
Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. The root directory of a process need not be the root directory of the root file system.

File Access Permissions.
Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches the user ID of the owner of the file and the appropriate access bit of the *owner* portion (0700) of the file mode is set.

The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process matches the group of the file and the appropriate access bit of the *group* portion (070) of the file mode is set.

The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process does not match the group ID of the file, and the appropriate access bit of the *other* portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

Message Queue Identifier
A message queue identifier (msqid) is a unique positive integer created by a *msgget*(2) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as *msqid_ds* and contains the following members:

```
struct  ipc_perm msg_perm; /* operation permission struct */
ushort  msg_qnum;          /* number of msgs on q */
ushort  msg_qbytes;        /* max number of bytes on q */
ushort  msg_lspid;         /* pid of last msgsnd operation */
ushort  msg_lrpid;         /* pid of last msgrcv operation */
time_t  msg_stime;         /* last msgsnd time */
time_t  msg_rtime;         /* last msgrcv time */
time_t  msg_ctime;         /* last change time */
                           /* times measured in secs since */
```

/* 00:00:00 GMT, Jan. 1, 1970 */

Msg_perm is a ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort  cuid;        /* creator user id */
ushort  cgid;        /* creator group id */
ushort  uid;         /* user id */
ushort  gid;         /* group id */
ushort  mode;        /* r/w permission */
```

Msg_qnum is the number of messages currently on the queue.

Msg_qbytes is the maximum number of bytes allowed on the queue.

Msg_lspid is the process id of the last process that performed a *msgsnd* operation.

Msg_lrpid is the process id of the last process that performed a *msgrcv* operation.

Msg_stime is the time of the last *msgsnd* operation.

Msg_rtime is the time of the last *msgrcv* operation.

Msg_ctime is the time of the last *msgctl*(2) operation that changed a member of the above structure.

Message Operation Permissions.

In the *msgrcv*(2), *msgsnd*(2), and *msgctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where {*token*} is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a msqid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **msg_perm.[c]uid** in the data structure associated with *msqid* and the appropriate bit of the *user* portion (0600) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process matches **msg_perm.[c]gid** and the appropriate bit of the *group* portion (060) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process does not match **msg_perm.[c]gid** and the appropriate bit of the *other* portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

See *config*(1M) for information on how to enable these system calls.

Semaphore Identifier

A semaphore identifier (semid) is a unique positive integer created by a *semget*(2) system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid_ds* and contains the following members:

```
struct  ipc_perm sem_perm; /* operation permission struct */
ushort  sem_nsems;         /* number of sems in set */
time_t  sem_otime;         /* last operation time */
time_t  sem_ctime;         /* last change time */
                           /* Times measured in secs since */
                           /* 00:00:00 GMT, Jan. 1, 1970 */
```

Sem_perm is a ipc_perm structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort  cuid;        /* creator user id */
ushort  cgid;        /* creator group id */
ushort  uid;         /* user id */
ushort  gid;         /* group id */
ushort  mode;        /* r/a permission */
```

Sem_nsems is the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a *sem_num*. Sem_num values run sequentially from 0 to the value of sem_nsems minus 1.

Sem_otime is the time of the last *semop*(2) operation.

Sem_ctime is the time of the last *semctl*(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort  semval;     /* semaphore value */
short   sempid;     /* pid of last operation  */
ushort  semncnt;    /* # awaiting semval > cval */
ushort  semzcnt;    /* # awaiting semval = 0 */
```

Semval is a non-negative integer. **Sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **Semncnt** is a count of the number of processes that are currently suspended awaiting the semval of this semaphore to become greater than its current value. **Semzcnt** is a count of the number of processes that are currently suspended awaiting the semval of this semaphore to become zero.

Semaphore Operation Permissions.

In the *semop*(2) and *semctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where *token* is the type of permission needed interpreted as follows:

00400	Read by user
00200	Alter by user
00060	Read, Alter by group

00006	Read, Alter by others

Read and Alter permissions on a semid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **sem_perm.[c]uid** in the data structure associated with *semid* and the appropriate bit of the *user* portion (0600) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process matches **sem_perm.[c]gid** and the appropriate bit of the *group* portion (060) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process does not match **sem_perm.[c]gid** and the appropriate bit of the *other* portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

See *config*(1M) for information on how to enable these system calls.

Shared Memory Identifier

A shared memory identifier (shmid) is a unique positive integer created by a *shmget*(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as *shmid_ds* and contains the following members:

```
struct   ipc_perm shm_perm; /* operation permission struct */
int      shm_segsz;         /* size of segment */
ushort   shm_cpid;          /* creator pid */
ushort   shm_lpid;          /* pid of last operation */
short    shm_nattch;        /* number of current attaches */
time_t   shm_atime;         /* last attach time */
time_t   shm_dtime;         /* last detach time */
time_t   shm_ctime;         /* last change time */
                            /* Times measured in secs since */
                            /* 00:00:00 GMT, Jan. 1, 1970 */
```

Shm_perm is a ipc_perm structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort   cuid;     /* creator user id */
ushort   cgid;     /* creator group id */
ushort   uid;      /* user id */
ushort   gid;      /* group id */
ushort   mode;     /* r/w permission */
```

Shm_segsz specifies the size of the shared memory segment.

Shm_cpid is the process id of the process that created the shared memory identifier.

Shm_lpid is the process id of the last process that performed a *shmop*(2) operation.

Shm_nattch is the number of processes that currently have this segment attached.

Shm_atime is the time of the last *shmat* operation.

Shm_dtime is the time of the last *shmdt* operation.

Shm_ctime is the time of the last *shmctl*(2) operation that changed one of the members of the above structure.

Shared Memory Operation Permissions.

In the *shmop*(2) and *shmctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where *token* is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a shmid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **shm_perm.[c]uid** in the data structure associated with *shmid* and the appropriate bit of the *user* portion (0600) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process matches **shm_perm.[c]gid** and the appropriate bit of the *group* portion (060) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process does not match **shm_perm.[c]gid** and the appropriate bit of the *other* portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

See *config*(1M) for information on how to enable these system calls.

ERRORS

All of the possible error numbers are not listed in each system call description because many errors are possible for most of the calls. The following is a complete list of the error numbers and their names as defined in **<errno.h>**. Some errors have several possible causes, noted in separate paragraphs following the error names below.

1 EPERM Not owner

Modification of a file is forbidden except to its owner or super-user.

The requested system call is allowed only to the super-user.

2 ENOENT No such file or directory

A specified file does not exist.

One of the directories in a path name does not exist.

3 ESRCH No such process

No process can be found corresponding to that specified by *pid* in *kill* or *ptrace*.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, has occurred during a system call. If execution is resumed after processing the signal, the interrupted system call appears to return this error condition.

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

6 ENXIO No such device or address
I/O on a special file refers to a subdevice which does not exist, or is beyond the limits of the device.
A tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 5,120 bytes is passed to an *exec* call.

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see *a.out*(4)).

9 EBADF Bad file number
A file descriptor refers to no open file,
A read (write) request is made to a file which is open only for writing (reading).

10 ECHILD No child processes
A *wait* was executed by a process that has no existing or unwaited-for child processes.

11 EAGAIN No more processes
A *fork* failed because the system process table is full or the user is not allowed to create any more processes.

12 ENOMEM Not enough space
During an *exec*, *brk*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter.
The arrangement of text, data, and stack segments requires too many segmentation registers.
There is not enough swap space during a *fork*.

13 EACCES Permission denied
The protection system forbids access to the file.

14 EFAULT Bad address
The system encountered a hardware fault in attempting to use an argument of a system call.

15 ENOTBLK Block device required
A non-block file is mentioned where a block device is required, e.g., in *mount*.

16 EBUSY Mount device busy
The device to be mounted is already mounted.
The device to be dismounted has an active file (open file, current directory, mounted-on file, active text segment).
Accounting is already enabled.

17 EEXIST File exists
An existing file is mentioned in an inappropriate context,

e.g., *link*.

18 EXDEV Cross-device link
A link to a file on another device is impossible.

19 ENODEV No such device
The system call is inappropriate for the device e.g., read a write-only device.

20 ENOTDIR Not a directory
A non-directory is specified where a directory is required, for example, in a path prefix or as an argument to *chdir*(2).

21 EISDIR Is a directory
A directory cannot be written to.

22 EINVAL Invalid argument
Some invalid argument (e.g., dismounting a non-mounted device; mentioning an undefined signal in *signal*, or *kill*; reading or writing a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (3M) entries of this manual.

23 ENFILE File table overflow
The system table of open files is full, and temporarily no more *opens* can be accepted.

24 EMFILE Too many open files
No process may have more than 64 file descriptors open at a time.

25 ENOTTY Not a typewriter
The device is not a typewriter or, in some cases, a block device is specified where a character device is required.

26 ETXTBSY Text file busy
The pure-procedure program to be executed is currently open for writing (or reading).
The pure-procedure program to be opened for writing is being executed.

27 EFBIG File too large
The size of a file exceeds the maximum file size (268,435,456,000 bytes) or ULIMIT; see *ulimit*(2).

28 ENOSPC No space left on device
During a *write* to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
A pipe cannot accept an *lseek*.

30 EROFS Read-only file system
A file or directory to be modified is on a device mounted read-only.

31 EMLINK Too many links
The maximum number of links (1000) to a file cannot be exceeded.

32 EPIPE Broken pipe
A pipe has no process to read the data being written. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Result too large
The value of a function in the math package (3M) is not representable within machine precision.
35 ENOMSG No message of desired type
A message to be received is of a type that does not exist on the specified message queue; see *msgop*(2).
36 EIDRM Identifier removed
The process is resuming execution due to the removal of an identifier from the file system name space (see *msgctl*(2), *semctl*(2), and *shmctl*(2)).
37 ECHRNG Channel number out of range
Not applicable to the system.
38 EL2NSYNC Level 2 non synchronized
Not applicable to the system.
39 EL3HLT Level 3 halted
Not applicable to the system.
40 EL3RST Level 3 reset
Not applicable to the system.
41 ELNRNG Link number out of range
Not applicable to the system.
42 EUNATCH Protocol driver not attached
Not applicable to the system.
43 ENOCSI No CSI structure available
Not applicable to the system.
44 EL2HLT Level 2 halted
Not applicable to the system.
45 EDEADLOCK Locking deadlock
A deadlock would occur if the lock were allowed.

SEE ALSO

config(1M), close(2), ioctl(2), open(2), pipe(2), read(2), write(2), intro(3).

SUPPORT STATUS

Supported.

NAME

access — determine accessibility of a file

SYNOPSIS

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

DESCRIPTION

Access checks the file specified by *path* for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. The bit pattern contained in *amode* is constructed as follows:

04	read
02	write
01	execute (search)
00	check existence of file

Path points to a path name naming a file.

If the user is the owner of the file, *access* checks the owner permissions. If the user (other than the owner) has group access to the file, *access* checks the group pemissions. *Access* checks other permissions for all other users.

FAILURE CONDITIONS

Access to the file is denied if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

Access requested read, write, or execute (search) permission for a null path name. [ENOENT]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

Access requested write access for a file on a read-only file system. [EROFS]

Access requested write access for a pure procedure (shared text) file that is being executed. [ETXTBSY]

Permission bits of the file mode do not permit the requested access. [EACCES]

Path points outside the allocated address space for the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

chmod(2), stat(2).

SUPPORT STATUS

Supported.

NAME

acct — enable or disable process accounting

SYNOPSIS

int acct (path)
char *path;

DESCRIPTION

Acct enables or disables the system process accounting routine. When enabled, the routine writes an accounting record on an accounting file for each process that terminates. One of two things terminate a process: an *exit* call or a signal; see *exit*(2) and *signal*(2).

The effective user ID of the calling process must be super-user to use this call.

ARGUMENTS

Path points to a path name naming the accounting file. The accounting file format is given in *acct*(4).

If *path* is non-zero and no errors occur during the system call, *acct* enables the accounting routine. If *path* is zero and no errors occur during the system call, *acct* disables the accounting routine.

FAILURE CONDITIONS

Acct fails if one or more of the following is true:

The effective user ID of the calling process is not super-user. [EPERM]

Acct attempts to enable accounting when it is already enabled. [EBUSY]

A component of the path prefix is not a directory. [ENOTDIR]

One or more components of the path name for the accounting file does not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

The file named by *path* is not an ordinary file. [EACCES]

Mode permission is denied for the named accounting file. [EACCES]

The named file is a directory. [EISDIR]

The named file resides on a read-only file system. [EROFS]

Path points to an invalid address. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

exit(2), signal(2), acct(4).

SUPPORT STATUS

Supported.

NAME

alarm – set the process alarm clock for a process

SYNOPSIS

unsigned alarm (sec)
unsigned sec;

DESCRIPTION

Alarm instructs the alarm clock for the calling process to send the signal SIGALRM to the calling process after the number of real time seconds specified by *sec* have elapsed; see *signal*(2).

Alarm requests are not stacked; successive calls reset the alarm clock for the calling process.

If *sec* is 0, *alarm* cancels any previous alarm request.

RETURN VALUE

Alarm returns the amount of time remaining in the alarm clock for the calling process *before* the request was made.

SEE ALSO

pause(2), signal(2).

SUPPORT STATUS

Supported.

NAME

brk, sbrk — change data segment space allocation

SYNOPSIS

int brk (endds)
char *endds;

char *sbrk (incr)
int incr;

DESCRIPTION

Brk and *sbrk* dynamically change the amount of space allocated for the data segment of the calling process; see *exec*(2). Both commands reset the *break value* for the process and allocate the appropriate amount of space. The *break value* is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the *break value* increases. Both commands set the newly allocated space to zero.

Brk sets the break value to *endds* and changes the allocated space accordingly.

Sbrk adds *incr* bytes to the break value and changes the allocated space accordingly. *Incr* can be negative, in which case the amount of allocated space decreases.

FAILURE CONDITIONS

Brk and *sbrk* fail without making any change in the allocated space if one or more of the following is true:

> Such a change would result in more space being allocated than is allowed by a system-imposed maximum (see *ulimit*(2)). [ENOMEM]
>
> Such a change would result in the break value being greater than or equal to the start address of any attached shared memory segment (see *shmop*(2)).

RETURN VALUE

old break value	successful completion (sbrk)
0	successful completion (brk)
-1	error; **errno** indicates the error

SEE ALSO

exec(2), shmop(2), ulimit(2).

SUPPORT STATUS

Supported.

NAME

chdir − change working directory

SYNOPSIS

int chdir (path)
char *path;

DESCRIPTION

Chdir changes the current working directory to the directory specified by *path*. *Path* points to the path name of a directory.

FAILURE CONDITIONS

Chdir fails and does not change the current working directory if one or more of the following is true:

A component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

Search permission is denied for any component of the path name. [EACCES]

Path points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

chroot(2).

SUPPORT STATUS

Supported.

NAME

chmod — change mode of file

SYNOPSIS

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

DESCRIPTION

Chmod sets the access permission portion of the mode for the file specified by *path* according to the bit pattern contained in *mode*. *Path* points to a path name naming a file.

Access permission bits are interpreted as follows:

04000	Set user ID on execution.
02000	Set group ID on execution.
01000	Save text image after execution
00400	Read by owner
00200	Write by owner
00100	Execute (or search if a directory) by owner
00070	Read, write, execute (search) by group
00007	Read, write, execute (search) by others

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

If the effective user ID of the process is not super-user, *chmod* clears mode bit 01000 (save text image on execution).

If the effective user ID of the process is not super-user or the effective group ID of the process does not match the group ID of the file, *chmod* clears mode bit 02000 (set group ID on execution).

If an executable file is prepared for sharing then mode bit 01000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time.

If the mode bit 02000 (set group ID on execution) is set and the mode bit 00010 (execute or search by group) is not set, mandatory file/record locking for writes is in effect for regular files locked using *lockf*(3X). This may effect future calls to *open*(2), *creat*(2), *read*(2), and *write*(2) on this file. Note that *lockf*(2) file/record locks are always enforced for writes to regular files.

FAILURE CONDITIONS

Chmod fails and does not change the file mode if one or more of the following is true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the allocated address space of a process. [EFAULT]

RETURN VALUE

0 successful completion

−1 error; **errno** indicates the error

SEE ALSO

chown(2), create(2), fcntl(2), lockf(2), lockf(3X), mknod(2), open(2), write(2).

SUPPORT STATUS

Supported.

NAME

chown — change owner and group of a file

SYNOPSIS

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

DESCRIPTION

Chown sets the owner ID and group ID of the named file to the numeric values contained in *owner* and *group* respectively. *Path* points to a path name naming a file.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If a user other than the super-user invokes *chown*, *chown* clears the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively.

FAILURE CONDITIONS

Chown fails and does not change the owner and group of the named file if one or more of the following is true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the allocated address space for the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

chmod(2), chown(1).

SUPPORT STATUS

Supported.

NAME

chroot – change root directory

SYNOPSIS

int chroot (path)
char *path;

DESCRIPTION

Chroot changes the root directory to the directory specified by *path*. *Path* points to a path name naming a directory.

The user's working directory is unaffected by the *chroot* system call.

The effective user ID of the process must be super-user to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. can not be used to access files outside the subtree rooted at the root directory.

FAILURE CONDITIONS

Chroot fails and does not change the root directory if one or more of the following is true:

> Any component of the path name is not a directory. [ENOENT]
>
> The named directory does not exist. [ENOENT]
>
> The effective user ID is not super-user. [EPERM]
>
> *Path* points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

chdir(2).

SUPPORT STATUS

Supported.

NAME

close — close a file descriptor

SYNOPSIS

int close (fildes)
int fildes;

DESCRIPTION

Close closes the file descriptor indicated by *fildes*, a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

FAILURE CONDITIONS

Close fails if *fildes* is not a valid open file descriptor. [EBADF]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2).

SUPPORT STATUS

Supported.

NAME

creat – create a new file or rewrite an existing one

SYNOPSIS

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

DESCRIPTION

Creat creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *path*.

If the file exists, *creat* truncates it to length 0, not changing the mode or owner. *Creat* sets the owner ID of the file to the effective user ID for the process, and sets the group ID of the file to the effective group ID for the process, and sets the low-order 12 bits of the file mode to the value of *mode* modified as follows:

> *Creat* clears all bits set in the file mode creation mask for the process. See *umask*(2).
>
> *Creat* clears the bit in the mode which saves the text image after execution. See *chmod*(2).

Upon successful completion, *creat* returns a non-negative integer, the file descriptor; the file is open for writing, even if the mode does not permit writing. The file pointer points to the beginning of the file. The file descriptor is set to remain open across *exec* system calls. See *fcntl*(2).

No process may have more than 64 files open simultaneously.

A new file may be created with a mode that forbids writing.

FAILURE CONDITIONS

Creat fails if one or more of the following is true:

> A component of the path prefix is not a directory. [ENOTDIR]
>
> A component of the path prefix does not exist. [ENOENT]
>
> Search permission is denied on a component of the path prefix. [EACCES]
>
> The path name is null. [ENOENT]
>
> The file does not exist and the directory in which the file is to be created does not permit writing. [EACCES]
>
> The named file resides or would reside on a read-only file system. [EROFS]
>
> The file is a pure procedure (shared text) file that is being executed. [ETXTBSY]
>
> The file exists and write permission is denied. [EACCES]
>
> The named file is an existing directory. [EISDIR]
>
> The maximum number of user file descriptors are currently open. [EMFILE]
>
> *Path* points outside the allocated address space for the process. [EFAULT]

The system file table is full. [ENFILE]

The file exists, *lockf*(2), *lockf*(3X), or *fcntl*(2) was used to lock a portion of the file, and mandatory enforcement mode is in effect for the file (see *chmod*(2), *lockf*(2), *lockf*(3X), or *fcntl*(2)). [EAGAIN]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the file descriptor
-1	error; *errno* indicates the error

SEE ALSO

chmod(2), close(2), dup(2), fcntl(2), lockf(3X), lockf(2), lseek(2), open(2), read(2), umask(2), write(2).

SUPPORT STATUS

Supported.

NAME

dup — duplicate an open file descriptor

SYNOPSIS

int dup (fildes)
int fildes;

DESCRIPTION

Dup returns a file descriptor having the following in common with *fildes*, a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call:

Same open file (or pipe).

Same file pointer. (i.e., both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

Dup sets the new file descriptor to remain open across *exec* system calls. See *fcntl*(2).

Dup returns the lowest file descriptor available.

FAILURE CONDITIONS

Dup fails if one or more of the following is true:

Fildes is not a valid open file descriptor. [EBADF]

Sixty four file descriptors are currently open. [EMFILE]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the file descriptor
−1	error; errno indicates the error

SEE ALSO

creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2).

SUPPORT STATUS

Supported.

NAME

execl, execv, execle, execve, execlp, execvp — execute a file

SYNOPSIS

```
int execl (path, arg0, arg1, ..., argn, 0)
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv[ ];

int execle (path, arg0, arg1, ..., argn, 0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

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

int execlp (file, arg0, arg1, ..., argn, 0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];
```

DESCRIPTION

Exec in all its forms transforms the calling process into a new process. *Exec* constructs the new process from an ordinary, executable file called the *new process file*. This file consists of a header (see *a.out*(4)), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). A successful *exec* does not return to the calling process because the new process overlays the calling process.

When a C program is executed, it is called as follows:

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

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. *Argc* is conventionally at least 1 and the first member of the array points to a string containing the name of the file.

ARGUMENTS

Path points to a path name that identifies the new process file.

File points to the new process file. *Exec* searches for the path prefix for this file in the directories passed as the *environment* line "PATH =" (see *environ*(5)). The shell supplies the environment (see *sh*(1)).

Arg0, *arg1*, ..., *argn* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, at least *arg0* must be present, and point to a string that is the same as *path* (or its last component).

Argv is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). A null pointer terminates *argv*.

Envp is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process. *Envp* is terminated by a null pointer. For *execl* and *execv*, the C run-time start-off routine places a pointer to the environment of the calling process in the global cell:

extern char **environ;

and that cell passes the environment of the calling process to the new process.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see *fcntl*(2). *Exec* does not change the file pointers for those file descriptors that remain open.

INHERITED ATTRIBUTES

Signals

Exec sets the signals of the new process as follows:

Signals set to terminate the calling process are set to terminate the new process.

Signals set to be ignored by the calling process are set to be ignored by the new process.

Signals set to be caught by the calling process are set to terminate new process; see *signal*(2).

Permission Modes

If the set-user-ID mode bit of the new process file is set (see *chmod*(2)), *exec* sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode bit of the new process file is set, *exec* sets the effective group ID of the new process to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

Shared Memory

Exec does not attach the shared memory segments attached to the calling process to the new process (see *shmop*(2)).

Profiling

Exec disables profiling for the new process; see *profil*(2).

Other

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

nice value (see *nice*(2))
process ID
parent process ID
process group ID
semadj values (see *semop*(2))
tty group ID (see *exit*(2) and *signal*(2))
trace flag (see *ptrace*(2) request 0)
time left until an alarm clock signal (see *alarm*(2))
current working directory
root directory

file mode creation mask (see *umask*(2))
file size limit (see *ulimit*(2))
utime, *stime*, *cutime*, and *cstime* (see *times*(2))

FAILURE CONDITIONS

Exec fails and returns to the calling process if one or more of the following is true:

One or more components of the path name for the new process file does not exist. [ENOENT]

A component of the path prefix to the new process file is not a directory. [ENOTDIR]

Search permission is denied for a directory listed in the path prefix to the new process file. [EACCES]

The new process file is not an ordinary file. [EACCES]

The new process file mode denies execution permission. [EACCES]

The *exec* command is not an *execlp* or *execvp*, and the new process file has the appropriate access permission but an invalid magic number in its header. [ENOEXEC]

The new process file is a pure procedure (shared text) file that is currently open for writing by some process. [ETXTBSY]

The new process requires more memory than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]

The number of bytes in the argument list of the new process is greater than the system-imposed limit of 5120 bytes. [E2BIG]

The new process file is not as long as indicated by the size values in its header. [EFAULT]

Path, *argv*, or *envp* point to an invalid address. [EFAULT]

RETURN VALUE

Exec returns to the calling process if an error has occurred; the return value is −1 and **errno** is set to indicate the error.

SEE ALSO

alarm(2), exit(2), fork(2), nice(2), ptrace(2), semop(2), signal(2), times(2), ulimit(2), umask(2), a.out(4), environ(5), sh(1).

SUPPORT STATUS

Supported.

NAME

exit, _exit — terminate process

SYNOPSIS

void exit (status)
int status;
void _exit (status)
int status;

DESCRIPTION

Exit terminates the calling process with the following consequences:

Exit closes all of the file descriptors open in the calling process.

If the parent process of the calling process is executing a *wait*, *exit* notifies the parent process of the termination of the calling process and passes the low order eight bits (i.e., bits 0377) of *status* to it; see *wait*(2).

If the parent process of the calling process is not executing a *wait*, *exit* transforms the calling process into a zombie process. A *zombie process* is a process that only occupies a slot in the process table; it has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see <sys/proc.h>) to be used by *times.*

Exit sets the parent process ID of all of the existing child processes of the calling process and zombie processes to 1. Thus, the initialization process (see *intro*(2)) inherits each of these processes.

Exit detaches each attached shared memory segment and decrements the value of **shm_nattach** in the data structure associated with its shared memory identifier by 1.

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

If the process has a process, text, or data lock, *exit* performs an *unlock* (see *plock*(2)).

Exit writes an accounting record on the accounting file if the system accounting routine is enabled; see *acct*(2).

If the process ID, tty group ID, and process group ID of the calling process are equal, *exit* sends the **SIGHUP** signal to each processes that has a process group ID equal to that of the calling process.

The C function *exit* may cause cleanup actions before the process exits. The function *_exit* circumvents all cleanup.

SEE ALSO

acct(2), intro(2), plock(2), semop(2), signal(2), wait(2).

WARNING

See *WARNING* in *signal*(2).

SUPPORT STATUS

Supported.

NAME

fcntl — file control

SYNOPSIS

#include <fcntl.h>

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

DESCRIPTION

Fcntl controls open files.

ARGUMENTS

Fildes is an open file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

The *cmds* available are:

F_DUPFD
: Return a new file descriptor as follows:

- Lowest numbered available file descriptor greater than or equal to *arg*.
- Same open file (or pipe) as the original file.
- Same file pointer as the original file (i.e., both file descriptors share one file pointer).
- Same access mode (read, write or read/write).
- Same file status flags (i.e., both file descriptors share the same file status flags).
- The close-on-exec flag associated with the new file descriptor is set to remain open across *exec*(2) system calls.

F_GETFD
: Get the close-on-exec flag associated with the file descriptor *fildes*. If the low-order bit is 0 the file is to remain open across *exec*, otherwise the file is to be closed upon execution of *exec*.

F_SETFD
: Set the close-on-exec flag associated with *fildes* to the low-order bit of *arg* (0 or 1 as for F_GETFD).

F_GETFL
: Get *file* status flags.

F_SETFL
: Set *file* status flags to *arg*. Only certain flags can be set; see *fcntl*(5).

F_GETLK
: Get the first lock which blocks the lock description given by the variable of type *struct flock* pointed to by *arg* . The information retreived overwrites the information passed to *fcntl* in the *flock* structure. If no lock is found that would prevent this lock from being created, then the structure is passed back unchanged except for the lock type which is set to F_UNLCK.

F_SETLK

Set or clear a file segment lock according to the variable of type *struct flock* pointed to by *arg* (see *fcntl*(5)). The *cmd* F_SETLK is used to establish read (F_RDLCK) and write (F_WRLCK) locks, as well as remove ethier type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined by the <*fcntl.h*> header file. If a read or write lock can not be set, *fcntl* returns with an error value of −1.

F_SETLKW

This *cmd* is the same as F_SETLK except that if a read or write lock is blocked by other locks, the process sleeps until the segment is free to be locked.

A read lock prevents any process from write locking the protected area. More than one read lock may exist for a given segment of a file at a given time. The file descriptor on which a read lock is being placed must have been opened with read access.

A write lock prevents any process from read locking or write locking the protected area. Only one write lock may exist for a given segment of a file at a given time. The file descriptor on which a write lock is placed must be opened with write access.

The structure *flock* defined by the <*fcntl.h*> header file describes a lock. It describes the type (*l_type*), starting offset (*l_whence*), reletive offset (*l_start*), size (*l_len*), and *process-ID*:

```
short l_type;    /* F_RDLCK, F_WRLCK, F_UNLCK */
short l_whence;  /* flag for starting offset */
long  l_start;   /* reletive offset in bytes */
long  l_len;     /* if 0 then until EOF */
short l_sysid;   /* unused */
short l_pid;     /* returned with F_GETLK */
```

The process id field is used only with the F_GETLK to return the values for a blocking lock. Locks may start and extend beyond the current end of a file, but may not be negetive reletive to the beginning of the file. A lock may be set to always entend to end of file by setting *l_len* to zero. I such a lock also has *l_start* set to zero, the whole file is locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process causes the old lock to be removed and the new lock type to take effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a *fork*(2) system call.

When mandatory file and record locking is active on a file (see *chmod*(2)), future *read* and *write* system calls on the file are affected by the record locks in effect.

FAILURE CONDITIONS

Fcntl fails if one or more of the following is true:

Fildes is not a valid open file descriptor. [EBADF]

Cmd is F_DUPFD and *arg* is either negetive or greater than or equal to the maximum number of open file descriptors for each user. [EMFILE]

Cmd is F_DUPFD and the maximum number of files open for a user is exceeded. [EINVAL]

Cmd is F_GETLK, F_SETLK, or F_SETLKW and *arg* or the data it points to is not valid. [EINVAL]

Cmd is F_SETLK , the type of lock (*l_type*) is a read (F_RDLCK) lock and the segment of a file to be locked is already write locked by another process or the type is a write (F_WRLCK) lock and the segment of a file to be locked is already read or write locked by another process. [EAGAIN]

Cmd is F_SETLK or F_SETLKW, the type of lock is a read or write lock, and there are no more record lock resources available. [ENOLCK]

Cmd is F_SETLKW, the lock is blocked by some lock from another process and sleeping (waiting) for that lock to become free. This would cause a deadlock situation. [EDEADLK]

RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

F_DUPFD	A new file descriptor.
F_GETFD	Value of flag (only the low-order bit is defined).
F_SETFD	Value other than −1.
F_GETFL	Value of file flags.
F_SETFL	Value other than −1.
F_GETLK	Value other than −1.
F_SETLK	Value other than −1.
F_SETLKW	Value other than −1.

Otherwise, *fcntl* returns a value of −1 and sets *errno* to indicate the error.

SEE ALSO

close(2), creat(2), dup(2), exec(2), fork(2), open(2), pipe(2), fcntl(5).

SUPPORT STATUS

Level 1. Supported.

NAME

ffp,sfp — floating point processor access

DESCRIPTION

Ffp permits access to the SKY Floating Point Processor for single or double precision add, subtract, multiply, and divide operations on data represented in IEEE 754 format. The assembler interface is:

```
movl #10xx, d0        ;SKY command
movl opr1, a0         ;address of first operand
movl opr2, a1         ;address of second operand
trap #0x0d            ;execute trap
movl d0, result       ;32 bit result
```

For double precision:

```
movl d1, result+4     ;second half of answer
```

Sfp is like *ffp* except that all functions of the processor are available, but the operation is slower. The assembler interface is:

```
movl #10xx, d0        ;SKY command
movl opr1, a0         ;address of first operand
movl opr2, a1         ;address of second operand
trap #0x0e            ;execute trap
movl d0, result       ;32 bit result
```

For double or complex:

```
movl d1, result+4     ;second half of answer
```

For function calls:

```
movl #10xx, d0        ;SKY command
movl opr1, a0         ;address of first operand
trap #0x0e            ;execute trap
movl d0, result       ;32 bit result
```

or

```
movl d1, result+4     ;second half of answer
```

SEE ALSO

skyload(1M) and SKY FFP System Integration Manual, SKY Computers.

SUPPORT STATUS

Not supported.

NAME

fork – create a new process

SYNOPSIS

int fork ()

DESCRIPTION

Fork creates a new process (child process), that is an exact copy of the calling process (parent process).

INHERITED ATTRIBUTES

The child process differs from the parent process in the following ways:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process has its own copy of the file descriptors of the parent. Each of the file descriptors of the child shares a common file pointer with the corresponding file descriptor of the parent.

Fork clears all semadj values (see *semop*(2)).

The child does not inherit process locks, text locks and data locks (see *plock*(2)).

Fork sets *utime*, *stime*, *cutime*, and *cstime* for the child process to 0. The time left until an alarm clock signal is reset to 0.

The child process inherits the following attributes from the parent process:

environment
close-on-exec flag (see *exec*(2))
signal handling settings (i.e., SIG_DFL, SIG_ING, function address)
set-user-ID mode bit
set-group-ID mode bit
profiling on/off status
nice value (see *nice*(2))
all attached shared memory segments (see *shmop*(2))
process group ID
tty group ID (see *exit*(2) and *signal*(2))
trace flag (see *ptrace*(2) request 0)
time left until an alarm clock signal (see *alarm*(2))
current working directory
root directory
file mode creation mask (see *umask*(2))
file size limit (see *ulimit*(2))

FAILURE CONDITIONS

Fork fails and does not create the child process if one or more of the following is true:

The system-imposed limit on the total number of processes under execution would be exceeded. [EAGAIN]

The system-imposed limit on the total number of processes under execution by a single user would be exceeded. [EAGAIN]

RETURN VALUE

successful completion
- —returns a value of 0 to the child process
- —returns the process ID of the child process to the parent process

unsuccessful completion
- —returns a value of −1 to the parent process
- —no child process is created

SEE ALSO

exec(2), times(2), wait(2).

SUPPORT STATUS

Supported.

NAME

getpid, getpgrp, getppid — get process, process group, and parent process IDs

SYNOPSIS

int getpid ()

int getpgrp ()

int getppid ()

DESCRIPTION

Getpid returns the process ID of the calling process.

Getpgrp returns the process group ID of the calling process.

Getppid returns the parent process ID of the calling process.

SEE ALSO

exec(2), fork(2), intro(2), setpgrp(2), signal(2).

SUPPORT STATUS

Supported.

NAME

getuid, geteuid, getgid, getegid — get real user, effective user, real group, and effective group IDs

SYNOPSIS

unsigned short getuid ()

unsigned short geteuid ()

unsigned short getgid ()

unsigned short getegid ()

DESCRIPTION

Getuid returns the real user ID of the calling process.

Geteuid returns the effective user ID of the calling process.

Getgid returns the real group ID of the calling process.

Getegid returns the effective group ID of the calling process.

SEE ALSO

intro(2), setuid(2).

SUPPORT STATUS

Supported.

NAME

ioctl — control device

SYNOPSIS

ioctl (fildes, request, arg)
int fildes, request;

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The pages describing the various devices in Section 7 of the Superuser Reference Manual discuss how *ioctl* applies to those devices.

FAILURE CONDITIONS

Ioctl fails if one or more of the following is true:

Fildes is not a valid open file descriptor. [EBADF]

Fildes is not associated with a character special device. [ENOTTY]

Request or *arg* is not valid. See Section 7 of the Superuser Reference Manual. [EINVAL]

RETURN VALUE

If an error has occurred, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

termio(7).

SUPPORT STATUS

Supported.

NAME

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

SYNOPSIS

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

DESCRIPTION

Kill sends a signal specified by *sig* to a process or a group of processes specified by *pid*. The specified signal is either one from the list given in *signal*(2), or 0. If *sig* is 0 (the null signal), *kill* performs error checking but does not actually send the signal. Thus the null signal can be used to check the validity of *pid*.

The real or effective user ID of the sending process must match the real or effective user ID of the receiving process, unless the effective user ID of the sending process is super-user.

ARGUMENTS

The processes with a process ID of 0 and a process ID of 1 are special processes (see *intro*(2)) and are referred to below as *proc0* and *proc1* respectively.

The value of *pid* affects the destination of *sig* as follows:

greater than 0
: send *sig* to the process whose process ID is *pid*. *Pid* may equal 1.

0
: send *sig* to all processes excluding *proc0* and *proc1* whose process group ID is equal to the process group ID of the sender.

−1
: if the effective user ID of the sender is not super-user, send *sig* to all processes excluding *proc0* and *proc1* whose real user ID is equal to the effective user ID of the sender. Otherwise, send *sig* to all processes excluding *proc0* and *proc1*.

negative, not −1
: send *sig* to all processes whose process group ID is equal to the absolute value of *pid*.

FAILURE CONDITIONS

Kill fails and sends no signal if one or more of the following is true:

Sig is not a valid signal number. [EINVAL]

Kill cannot find a process corresponding to the specified *pid*. [ESRCH]

The user ID of the sending process is not super-user, and its real or effective user ID does not match the real or effective user ID of the receiving process. [EPERM]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

getpid(2), setpgrp(2), signal(2), kill(1).

SUPPORT STATUS

Supported.

NAME

link — link to a file

SYNOPSIS

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

DESCRIPTION

Link creates a new link (directory entry) for the existing file.

ARGUMENTS

Path1 points to a path name naming an existing file. *Path2* points to a path name naming the new directory entry to be created.

FAILURE CONDITIONS

Link fails and does not create a link if one or more of the following is true:

A component of either path prefix is not a directory. [ENOTDIR]

A component of either path prefix does not exist. [ENOENT]

A component of either path prefix denies search permission. [EACCES]

The file named by *path1* does not exist. [ENOENT]

The link named by *path2* exists. [EEXIST]

The file named by *path1* is a directory and the effective user ID is not super-user. [EPERM]

The link named by *path2* and the file named by *path1* are on different logical devices (file systems). [EXDEV]

Path2 points to a null path name. [ENOENT]

The requested link requires writing in a directory with a mode that denies write permission. [EACCES]

The requested link requires writing in a directory on a read-only file system. [EROFS]

Path points outside the allocated address space of the process. [EFAULT]

The maximum number of links is exceeded. [EMLINK]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

unlink(2).

SUPPORT STATUS

Supported.

NAME

lockf, locking — provide exclusive file regions for reading and writing

SYNOPSIS

locking(fildes, mode, size)
lockf(fildes, mode, size)
int fildes;
int mode;
int size;

DESCRIPTION

NOTE: This system call is the TOWER compatible call. See *lockf*(3X) for the standard UNIX system call.

Lockf (also known as *locking*) allows a specified number of bytes to be accessed only by the locking process. Other processes which attempt to lock, read, or write the locked area sleep until the area becomes unlocked (Mandatory write locking is enforced).

Fildes is the word returned from a successful *open*, *creat*, *dup*, or *pipe* system call.

Mode is zero to unlock an area. *Mode* is one or two to lock an area. If the *mode* is one, and the area has some other lock on it, the process sleeps until the area is available. If the *mode* is two, and the area is locked, an error is returned.

Size is the number of contiguous bytes to be locked or unlocked. The area to be locked starts at the current offset in the file. If *size* is zero, the area to the end of file is locked.

Because the potential for a deadlock occurs when the process controlling a locked area is put to sleep be accessing the locked area of another process, calls to *lockf*, *read*, or *write* scan for a deadlock prior to sleeping on a locked area. An error return is made if sleeping on the locked area would cause a deadlock.

Lock requests may, in whole or part, contain or be contained by a previously locked area for the same process. When this or adjacent areas occur, the areas are combined into a single area. If the request requires a new lock element with the lock table full, an error is returned and the area is not locked.

Unlock requests may, in whole or part, release one or more locked regions controlled by the process. When regions are not fully released, the remaining areas are still locked by the process. Release of the center section of a locked area requires an additional lock element to hold the cutoff section. If the lock table full, an error is returned and the area is not released.

While locks may be applied to special files or pipes, read/write operations are not blocked. Locks may not be applied to a directory.

Both *lockf*(2) and lockf (3X) type locking calls can be issued concurrently, with the exception that all locking requests for a given file must be of the same type. An error exit is taken with errno set to EBUSY if a lock request is made to a file that already has existing locks of the other type.

FAILURE CONDITIONS

Below are listed error conditions with the corresponding *errno* returned :

[EBADF]
Fildes is not a valid open descriptor, or the locking mode and file open modes do not agree.

[EINTR]
The value of *mode* is not valid.

[EACCES]
Cmd is 2 and the section is already locked by another process.

[EDEADLK]
Cmd is 1 or 2 and a deadlock would occur. Also the *cmd* is either of the above or 0 and the number of entries in the lock table would exceed the number allocated on the system.

[EBUSY]
Any *cmd* issued against a file that already has locks of the wrong type.

WARNING

Note that *lockf*(2) and *lockf*(3X) calls, while being almost interface compatible are not functionally compatible unless the file permissions are set for mandatory lock enforcements on *write*(2) calls when using *lockf*(see *access*(2)). *lockf*(2) always enforces locks on *write*(2) call for regular files.

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

The obsolete error code define EDEADLOCK has the same value as EDEADLK. Users are encouraged to use EDEADLK for future upward compatibility.

This system call is only available when using the *cc20t* Source Generation Software.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

close(2), creat(2), fcntl(2), intro(2), lockf(3X), open(2), read(2), write(2).

SUPPORT STATUS

Not supported.

NAME

lseek — move read/write file pointer

SYNOPSIS

long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;

DESCRIPTION

Lseek sets the file pointer associated with *fildes* according to the value of *whence* as follows:

0	Set the pointer to *offset* bytes.
1	Set the pointer to its current location plus *offset*.
2	Set the pointer to the size of the file plus *offset*.

Upon successful completion, *lseek* returns the resulting pointer location as measured in bytes from the beginning of the file.

Fildes is a file descriptor returned from a *creat*, *open*, *dup*, or *fcntl* system call.

FAILURE CONDITIONS

Lseek fails and does not change the file pointer if one or more of the following is true:

Fildes is not an open file descriptor. [EBADF]

Fildes is associated with a pipe or fifo. [ESPIPE]

Whence is not 0, 1 or 2. [EINVAL and SIGSYS signal]

The resulting file pointer would be negative. [EINVAL]

WARNING

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

RETURN VALUE

non-negative integer	successful completion; the non-negative integer indicates the file pointer value
−1	error; errno indicates the error

SEE ALSO

creat(2), dup(2), fcntl(2), open(2).

SUPPORT STATUS

Supported.

NAME

mknod – make a directory, or a special or ordinary file

SYNOPSIS

int mknod (path, mode, dev)
char *path;
int mode, dev;

DESCRIPTION

Mknod creates a new file named by the path name pointed to by *path*. *Mknod* initializes the mode of the new file from *mode*, where the value of *mode* is interpreted as follows:

0170000 file type; one of the following:
 0010000 fifo special
 0020000 character special
 0040000 directory
 0060000 block special
 0100000 or 0000000 ordinary file

0004000 set user ID on execution

0002000 set group ID on execution

0001000 save text image after execution

0000777 access permissions; constructed from the following
 0000400 read by owner
 0000200 write by owner
 0000100 execute (search on directory) by owner
 0000070 read, write, execute (search) by group
 0000007 read, write, execute (search) by others

Mknod sets the owner ID of the file to the effective user ID of the process. *Mknod* also sets the group ID of the file to the effective group ID of the process.

Values of *mode* other than those above are undefined and should not be used.

The low-order 9 bits of *mode* are modified by the file mode creation mask of the process: *mknod* clears all bits set in the file mode creation mask of the process. See *umask*(2).

If *mode* indicates a block or character special file, *dev* is a configuration dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored.

Mknod may be invoked only by the super-user for file types other than FIFO special.

FAILURE CONDITIONS

Mknod fails and does not create the new file if one or more of the following is true:

The effective user ID of the process is not super-user. [EPERM]

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

The directory in which the file is to be created is located on a read-only file system. [EROFS]

The named file exists. [EEXIST]

Path points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

chmod(2), exec(2), umask(2), fs(4), mkdir(1).

SUPPORT STATUS

Supported.

NAME

mount — mount a file system

SYNOPSIS

int mount (spec, dir, rwflag)
char *spec, *dir;
int rwflag;

DESCRIPTION

Mount mounts a removable file system, contained on the block special file identified by *spec*, on the directory identified by *dir*. *Spec* and *dir* are pointers to path names.

Upon successful completion, references to the file *dir* refer to the root directory on the mounted file system.

The low-order bit of *rwflag* controls write permission on the mounted file system: if 1, writing is forbidden; if 0, writing is permitted according to individual file accessibility.

Only the super-user may invoke *mount*.

FAILURE CONDITIONS

Mount fails if one or more of the following is true:

The effective user ID is not super-user. [EPERM]

Any of the named files does not exist. [ENOENT]

A component of a path prefix is not a directory. [ENOTDIR]

Spec is not a block special device. [ENOTBLK]

The device associated with *spec* does not exist. [ENXIO]

Dir is not a directory. [ENOTDIR]

Spec or *dir* points outside the allocated address space of the process. [EFAULT]

A file system is currently mounted on *dir*. [EBUSY]

Dir is the current working directory for another user, or is otherwise busy. [EBUSY]

The device associated with *spec* is currently mounted. [EBUSY]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

umount(2).

SUPPORT STATUS

Supported.

NAME

msgctl − message control operations

SYNOPSIS

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

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

DESCRIPTION

Msgctl provides a variety of message control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro*(2). {READ}

IPC_SET Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

msg_perm.uid
msg_perm.gid
msg_perm.mode /* only low 9 bits */
msg_qbytes

This *cmd* can only be executed by a process that has an effective user ID equal to either that of the super-user or to the value of **msg_perm.uid** in the data structure associated with *msqid*. Only the super-user can raise the value of **msg_qbytes**.

IPC_RMID Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it.

This *cmd* can only be executed by a process that has an effective user ID equal to either that of the super-user or to the value of **msg_perm.uid** in the data structure associated with *msqid*.

FAILURE CONDITIONS

Msgctl fails if one or more of the following is true:

Msqid is not a valid message queue identifier. [EINVAL]

Cmd is not a valid command. [EINVAL]

Cmd is equal to IPC_STAT and {READ} operation permission is denied to the calling process (see *intro*(2)). [EACCES]

Cmd is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of the super-user and it is not equal to the value of **msg_perm.uid** in the data structure associated with *msqid*. [EPERM]

Cmd is equal to IPC_SET, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process is not equal to that of the super-user. [EPERM]

Buf points to an illegal address. [EFAULT]

RETURN VALUE

0 successful completion

−1 error; errno indicates the error

SEE ALSO

intro(2), msgget(2), msgop(2).

SUPPORT STATUS

Supported.

NAME

msgget – get message queue

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key, msgflg)
key_t key;
int msgflg;

DESCRIPTION

Msgget returns the message queue identifier associated with *key*.

Msgget creates a message queue identifier and associated message queue and data structure (see *intro*(2)) for *key* if one of the following is true:

Key is equal to IPC_PRIVATE.

Key does not already have a message queue identifier associated with it, and (*msgflg* & IPC_CREAT) is *true*.

Msgget initializes the data structure associated with the new message queue identifier as follows:

Sets **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** to the effective user ID and effective group ID, respectively, of the calling process.

Sets the low-order 9 bits of **msg_perm.mode** to the low-order 9 bits of *msgflg*.

Sets **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** to 0.

Sets **msg_ctime** to the current time.

Sets **msg_qbytes** to the system limit.

FAILURE CONDITIONS

Msgget fails if one or more of the following is true:

A message queue identifier exists for *key* but operation permission (see *intro*(2)) as specified by the low-order 9 bits of *msgflg* is not granted. [EACCES]

A message queue identifier does not exist for *key* and (*msgflg* & IPC_CREAT) is *false*. [ENOENT]

The system imposed limit on the maximum number of allowed message queue identifiers on the system would be exceeded if the message queue identifier were created as requested. [ENOSPC]

A message queue identifier exists for *key* but ((*msgflg* & IPC_CREAT) & (*msgflg* & IPC_EXCL)) is *true*. [EEXIST]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the message queue identifier.

−1	error; **errno** indicates the error

SEE ALSO

intro(2), msgctl(2), msgop(2), msgsnd(2), msgrcv(2).

SUPPORT STATUS

Supported.

NAME

msgsnd, msgrcv — message send and receive operations

SYNOPSIS

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

int msgsnd (msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

DESCRIPTION

Msgsnd

Msgsnd is used to send a message to the queue associated with the message queue identifier specified by *msqid*. *Msgp* points to a structure containing the message. This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[];    /* message text */
```

Mtype is a positive integer that can be used by the receiving process for message selection (see *msgrcv* below). *Mtext* is any text of length *msgsz* bytes. *Msgsz* can range from 0 to a system imposed maximum.

Msgflg specifies the action to be taken if one or more of the following are true:

- The number of bytes already on the queue is equal to **msg_qbytes** (see *intro*(2)).
- The total number of messages on all queues system wide is equal to the system imposed limit.

These actions are as follows:

- If (*msgflg* & **IPC_NOWAIT**) is true, the message is not sent and the calling process returns immediately.
- If (*msgflg* & **IPC_NOWAIT**) is false, the calling process suspends execution until one of the following occurs:
  - The condition responsible for the suspension no longer exists, in which case the message is sent.
  - *Msqid* is removed from the system (see *msgctl*(2)). When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.
  - The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in *signal*(2)).

Msgsnd fails and no message is sent if one or more of the following are true:

- *Msqid* is not a valid message queue identifier. [EINVAL]
- Operation permission is denied to the calling process (see *intro*(2)). [EACCES]
- *Mtype* is less than 1. [EINVAL]
- The message cannot be sent for one of the reasons cited above and (*msgflg* & **IPC_NOWAIT**) is **true**. [EAGAIN]
- *Msgsz* is less than zero or greater than the system imposed limit. [EINVAL]
- *Msgp* points to an invalid address. [EFAULT]

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see intro (2)).
- **Msg_qnum** is incremented by 1.
- **Msg_lspid** is set equal to the process ID of the calling process.
- **Msg_stime** is set equal to the current time.

Msgrcv

Msgrcv reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[];    /* message text */
```

Mtype is the received message's type as specified by the sending process. *Mtext* is the text of the message. *Msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & **MSG_NOERROR**) is **true**. The truncated part of the message is lost and no indication of the truncation is given to the calling process.

Msgtyp specifies the type of message requested as follows:
- If *msgtyp* is equal to 0, the first message on the queue is received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

Msgflg specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:
- If (*msgflg* & **IPC_NOWAIT**) is **true**, the calling process returns immediately with a return value of −1 and **errno** set to ENOMSG.
- If (*msgflg* & **IPC_NOWAIT**) is **false**, the calling process suspends execution until one of the following occurs:
  - A message of the desired type is placed on the queue.
  - *Msqid* is removed from the system. When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.
  - The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in *signal*(2)).

Msgrcv fails and no message is received if one or more of the following are true:

- *Msqid* is not a valid message queue identifier. [EINVAL]
- Operation permission is denied to the calling process. [EACCES]
- *Msgsz* is less than 0. [EINVAL]
- *Mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is false. [E2BIG]
- The queue does not contain a message of the desired type and (*msgtyp* & IPC_NOWAIT) is true. [ENOMSG]
- *Msgp* points to an invalid address. [EFAULT]

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see *intro*(2)).

- Msg_qnum is decremented by 1.
- Msg_lrpid is set equal to the process ID of the calling process.
- Msg_rtime is set equal to the current time.

RETURN VALUES

If *msgsnd* or *msgrcv* return due to the receipt of a signal, a value of −1 is returned to the calling process and **errno** is set to EINTR. If they return due to removal of *msqid* from the system, a value of −1 is returned and **errno** is set to EIDRM.

Upon successful completion, the return value is as follows:

- *Msgsnd* returns a value of 0.
- *Msgrcv* returns a value equal to the number of bytes actually placed into *mtext*.

Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

SEE ALSO

intro(2), msgctl(2), msgget(2), signal(2).

SUPPORT STATUS

Supported.

NAME

nice — change priority of a process

SYNOPSIS

int nice (incr)
int incr;

DESCRIPTION

Nice adds the value of *incr* to the nice value of the calling process. The *nice value* of a process is a positive number; a larger nice value indicates a lower CPU priority.

The maximum nice value is 39 and the minimum nice value is 0. If the user requests values above or below these limits, the system sets the nice value to the corresponding limit.

FAILURE CONDITIONS

Nice fails and does not change the nice value if the following is true:

incr is negative or greater than 40

the effective user ID of the calling process is not a super-user [EPERM]

RETURN VALUE

integer	successful completion; *nice* returns the new nice value minus 20
−1	error; errno indicates the error

SEE ALSO

nice(1), exec(2).

SUPPORT STATUS

Supported.

NAME

open – open for reading or writing

SYNOPSIS

#include <fcntl.h>
int open (path, oflag [, mode])
char *path;
int oflag, mode;

DESCRIPTION

Open opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*.

Upon successful completion a non-negative integer, the file descriptor, is returned.

Open sets the file pointer used to mark the current position within the file to the beginning of the file.

Open sets the new file descriptor to remain open across *exec* system calls. See *fcntl*(2).

Note that no process may have more than 64 file descriptors open simultaneously.

ARGUMENTS

Path points to a path name naming a file.

Oflag values are constructed by *or*-ing flags from the following lists:

Only one of the following three flags may be used:

O_RDONLY
Open for reading only.

O_WRONLY
Open for writing only.

O_RDWR Open for reading and writing.

Any number of the following flags may be *or*-ed into the *oflag*.

O_NDELAY
This flag may affect subsequent reads and writes. See *read*(2) and *write*(2).

When opening a FIFO:

If O_NDELAY is set and O_RDONLY is set, an *open* returns without delay.

If O_NDELAY is set and O_WRONLY is set, an *open* returns an error if no process currently has the file open for reading.

If O_NDELAY is clear and O_RDONLY is set, an *open* blocks until a process opens the file for writing.

If O_NDELAY is clear and O_WRONLY is set, an *open* blocks until a process opens the file for reading.

When opening a file associated with a communication line:

If O_NDELAY is set the *open* returns without waiting for carrier.

If O_NDELAY is clear the *open* blocks until carrier is present.

O_APPEND
If set, *open* sets the file pointer to the end of the file prior to each write.

O_CREAT If the file exists, this flag has no effect. Otherwise, *open* creates the file and does the following:

Sets the owner ID of the file to the effective user ID of the process

Sets the group ID of the file to the effective group ID of the process

Sets the low-order 12 bits of the file mode are set to the value of *mode* with the following modifications (see *creat*(2)):

clear all bits set in the file mode creation mask. See *umask*(2).

clear the *save text image after execution bit* of the mode. See *chmod*(2).

O_TRUNC If the file exists, truncate its length to 0, and do not change the mode and owner.

O_EXCL If O_EXCL and O_CREAT are set, *open* fails if the file exists.

O_SYNC If set, all subsequent writes are synchronous. Typically, control is returned to the program that performs a write as soon as the data has been copied into a buffer by the operating system. The actual write to the disk or tape can be delayed for many seconds. The synchronous write waits until the data and all necessary *inode* information has been written to the disk before returning control to the writing program.

FAILURE CONDITIONS

Open fails and does not open the named file if one or more of the following is true:

A component of the path prefix is not a directory. [ENOTDIR]

O_CREAT is not set and the named file does not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

Oflag permission is denied for the named file. [EACCES]

The named file is a directory and *oflag* is write or read/write. [EISDIR]

The named file resides on a read-only file system and *oflag* is write or read/write. [EROFS]

The maximum (64) number of file descriptors are currently open. [EMFILE]

The named file is a character special or block special file, and the device associated with this special file does not exist. [ENXIO]

The file is a pure procedure (shared text) file that is being executed and *oflag* is write or read/write. [ETXTBSY]

Path points outside the allocated address space of the process. [EFAULT]

O_CREAT and O_EXCL are set, and the named file exists. [EEXIST]

O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading. [ENXIO]

A signal was caught during the *open* system call. [EINTR]

The system file table is full. [ENFILE]

The file exists, *lockf*(2), *lockf*(3X) or *fcntl*(2) was used to lock a portion of the file, and mandatory enforcement mode is in effect for the file (see *chmod*(2), *lockf*(2), *lockf*(3X), or *fcntl*(2)). [EAGAIN]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the file descriptor
−1	error; **errno** indicates the error

SEE ALSO

chmod(2), close(2), creat(2), dup(2), fcntl(2), lockf(3X), lockf(2), lseek(2), read(2), umask(2), write(2).

SUPPORT STATUS

Supported.

NAME

pause − suspend process until signal

SYNOPSIS

pause ()

DESCRIPTION

Pause suspends the calling process until the process receives a signal, which the calling process has not set to be ignored.

If the signal terminates the calling process, *pause* does not return.

If the signal is *caught* by the calling process and control returns from the signal catching-function (see *signal*(2)), the calling process resumes execution from the point of suspension with a return value of −1 from *pause* and *errno* set to EINTR.

SEE ALSO

alarm(2), kill(2), signal(2), wait(2).

SUPPORT STATUS

Supported.

NAME

pipe — create an interprocess channel

SYNOPSIS

int pipe (fildes)
int fildes[2];

DESCRIPTION

Pipe creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *Fildes*[0] is opened for reading and *fildes*[1] is opened for writing.

The pipe buffers writes up to 5120 bytes of data before blocking the writing process. A read on file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first-in-first-out basis.

No process may have more than 64 file descriptors open simultaneously.

FAILURE CONDITIONS

Pipe fails if 63 or more file descriptors are currently open. [EMFILE]

The system file table is full. [ENFILE]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

read(2), write(2), sh(1).

SUPPORT STATUS

Supported.

NAME
plock – lock process, text, or data in memory

SYNOPSIS
#include <sys/lock.h>

int plock (op)
int op;

DESCRIPTION
Plock locks the text segment (text lock), the data segment (data lock), or both the text and data segments (process lock) of the calling process into memory. Locked segments are immune to all routine swapping. *Plock* also unlocks these segments.

The effective user ID of the calling process must be super-user to use this call.

ARGUMENTS
Op specifies the following:

PROCLOCK
– lock text & data segments into memory (process lock)

TXTLOCK – lock text segment into memory (text lock)

DATLOCK – lock data segment into memory (data lock)

UNLOCK – remove locks

FAILURE CONDITIONS
Plock fails and does not perform the requested operation if one or more of the following is true:

The effective user ID of the calling process is not super-user. [EPERM]

Op is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process. [EINVAL]

Op is equal to **TXTLOCK** and a text lock, or a process lock already exists on the calling process. [EINVAL]

Op is equal to **DATLOCK** and a data lock, or a process lock already exists on the calling process. [EINVAL]

Op is equal to **UNLOCK** and no type of lock exists on the calling process. [EINVAL]

RETURN VALUE
0 successful completion
−1 error; **errno** indicates the error

SEE ALSO
exec(2), exit(2), fork(2).

SUPPORT STATUS
Supported.

NAME

profil − execution time profile

SYNOPSIS

void profil (buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;

DESCRIPTION

Profil examines the program counter (pc) of the user each clock tick (60th second), subtracts *offset* from it and multiplies the result by *scale*. If the resulting number corresponds to a word inside *buff*, *profil* increments that word. *Buff* points to an area of core whose length (in bytes) is given by *bufsiz*.

Profil interprets *scale* as an unsigned, fixed-point fraction with binary point at the left: 0177777 (octal) gives a 1-1 mapping of program counters to words in *buff*; 077777 (octal) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of *buff* (producing a non-interrupting core clock).

A *scale* of 0 or 1 turns off profiling. A *bufsiz* of 0 renders profiling ineffective. Profiling stops when an *exec* is executed, but remains on in child and parent both after a *fork*. Profiling stops if an update in *buff* would cause a memory fault.

RETURN VALUE

Not defined.

SEE ALSO

monitor(3C), prof(1).

SUPPORT STATUS

Supported.

NAME

ptrace — process trace

SYNOPSIS

int ptrace (request, pid, addr, data);
int request, pid, addr, data;

DESCRIPTION

Using *ptrace*, a parent process can control the execution of a child process. The child process executes normally until it encounters a signal; the process then stops, entering a *stopped state*. *Wait*(2) notifies the parent process. The parent process may then examine and modify the *core image* of the child process. After examination, the parent may terminate or continue the child process, and specify whether the child process is to ignore the signal that stopped it.

The *request* argument determines the precise function of the *ptrace* call, and assumes the following numeric values;

The child process must issue the following request if it is to be traced by its parent. If the parent does not expect to trace the child process, unexpected results may occur.

0 Turn on the trace flag of the child process. The flag indicates whether the child process should remain in the *stopped state* upon receipt of a signal, or the state specified by *func*; see *signal*(2). *Ptrace* ignores the *pid*, *addr*, and *data* arguments, and does not have a defined return value.

The remainder of the requests may only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

1, 2 Return the word at location *addr* in the address space of the child to the parent process. Request 1 and request 2 have the same results.

Ptrace ignores the *data* argument.

These two requests fail if *addr* is not the start address of a word, in which case *ptrace* returns a value of −1 to the parent process and sets *errno* to EIO.

3 Return to the parent process the word at location *addr* in the USER area of the child process in the system address space (see <sys/user.h>). Addresses in this area range from 0 to 2048.

Ptrace ignores the *data* argument.

This request fails if *addr* is not the start address of a word or is outside the USER area, in which case *ptrace* returns a value of −1 to the parent process and sets *errno* to EIO.

4, 5 Write the value given by the *data* argument into the address space of the child at location *addr*.

Request 4 and request 5 may be used with equal results.

These two requests fail if *addr* is a location in a pure procedure space and another process is executing in that space, or *addr* is not the start address of a word.

Upon successful completion, *ptrace* returns the value written into the address space of the child to the parent. Upon failure *ptrace* returns a value of −1 to the parent process and sets *errno* to EIO.

6 Write a few entries in the USER area of the child process. *Data* is the value that is to be written and *addr* is the location of the entry. The few entries that can be written are:

the general registers (i.e., registers 0−11 on the 3B20S, registers 0−7 on PDP-11s, and registers 0−15 on the VAX)

the condition codes of the Processor Status Word on the TOWER.

7 Resume execution of the child process.

If the *data* argument is 0, *ptrace* cancels all pending signals (including the one that caused the child to stop) before the child process resumes execution.

If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal; *ptrace* cancels any other pending signals.

The *addr* argument must be equal to 1 for this request.

Upon successful completion, *ptrace* returns the value of *data* to the parent. This request fails if *data* is not 0 or a valid signal number, in which case *ptrace* returns a value of −1 to the parent process and sets *errno* to EIO.

8 Terminate the child with the same consequences as *exit*(2).

9 Set the trace bit in the Processor Status Word of the child and then execute the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.

On the 3B20S there is no trace bit and this request returns an error.

Note that the trace bit remains set after an interrupt on PDP-11s but is turned off after an interrupt on the VAX.

To forestall possible fraud, *ptrace* inhibits the set-user-id facility on subsequent *exec*(2) calls. If a traced process calls *exec*, it will stop before executing the first instruction of the executed program showing signal **SIGTRAP**.

GENERAL ERRORS

Ptrace fails in general if one or more of the following are true:

Request is an illegal number. [EIO]

Pid identifies a child that does not exist or has not executed a *ptrace* with request 0. [ESRCH]

SEE ALSO

exec(2), signal(2), wait(2), sdb(1).

SUPPORT STATUS

Supported.

NAME

pwrnote — power recovery notification

SYNOPSIS

pwrnote(option)
int option;

DESCRIPTION

Pwrnote turns the notification of power fail/recovery on or off. The default is no notification of power fail/recovery. If *option* is 0 then power notification is on whereas a 1 turns off power notification.

Pwrnote works in conjunction with *signal*(2). To receive the SIGPWR signal it is necessary to call *pwrnote* with the option set to zero. The power fail/recovery notification option remains set until program termination or the *pwrnote* function is explicitly called with the option set to one.

A typical usage of the *pwrnote* function is in screen editors where it is necessary to repaint the terminal screen after power is restored.

DIAGNOSTIC(S)

An unknown option sets the external variable *errno* to EFAULT.

RESTRICTIONS

Pwrnote requires a functional battery backup unit in order to be of any use.

SEE ALSO

signal(2).

SUPPORT STATUS

Supported.

NAME

pwrtime — power recovery interval to single-user state

SYNOPSIS

#include <sys/types.h>

pwrtime(interval)
time_t interval;

DESCRIPTION

Pwrtime sets the time interval allowed for the system to return to its original multi-user state after a power failure occured. If power returns after *interval* seconds *init*(1M) will terminate all logged-in terminal processes bringing the operating system to the single-user state. Users will not be logged-out when the power failure was of a duration less than *interval* seconds. The default interval is 300 seconds (five minutes).

Only the super-user is allowed to execute *pwrtime.*

DIAGNOSTIC(S)

Usage of *pwrtime* by users other than the super-user sets the external variable *errno* to EACCES.

RESTRICTIONS

Pwrtime requires a functional battery backup unit in order to be of any use.

SEE ALSO

init(1M).

SUPPORT STATUS

Supported.

NAME

read – read from file

SYNOPSIS

int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION

Read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*. *Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

On devices capable of seeking, the *read* starts at a position in the file given by the file pointer associated with *fildes*. *Read* increments the file pointer by the number of bytes actually read before returning.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see *ioctl*(2) and *termio*(7)), or if the number of bytes left in the file is less than *nbyte* bytes. *Read* returns a value of 0 when an end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

> If O_NDELAY is set, the read returns a 0.
>
> If O_NDELAY is clear, the read blocks until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that currently has no data available:

> If O_NDELAY is set, the read returns a 0.
>
> If O_NDELAY is clear, the read blocks until data becomes available.

FAILURE CONDITIONS

Read fails if one or more of the following is true:

> *Fildes* is not a valid file descriptor open for reading. [EBADF]
>
> *Buf* points outside the allocated address space. [EFAULT]
>
> A signal was caught during the *read* system call. [EINTR]

RETURN VALUE

non-negative integer	successful completion; the non-negative number indicates the number of bytes actually read
−1	error; errno indicates the error

SEE ALSO

creat(2), dup(2), fcntl(2), ioctl(2), open(2), pipe(2), termio(7).

SUPPORT STATUS

Supported.

NAME

rmnt, urmnt – mount and unmount directorys across file systems

SYNOPSIS

int rmnt (directory1, directory2)
char *directory1, *directory2;

int urmnt (directory1)
char *directory1;

DESCRIPTION

Rmnt announces to the system that the subtree of *directory2* is to be mounted under *directory1*. (*Rmnt* is similar to the mount command). Both *directory2* and *directory1* must already exist. Following the rmnt call, the subtree of *directory2* is now logically under *directory1*; the old contents of *directory1* are inaccessible while *directory2* is rmounted. *Rmnt* may be invoked only by the superuser.

Urmnt announces to the system that *directory1* is no longer rmounted. It is similar to the umount command. *Directory1* must exist already; *urmnt* restores subtree of *directory1* to the state it was before the *rmnt*(1M) was issued. *Urmnt* may be invoked only by the superuser.

FAILURE CONDITIONS

Rmnt fails and does not create a mount if one or more of the following is true:

> *Directory2* or *directory1* is not a directory. [ENOTDIR]
>
> *Directory2* or *directory1* does not exist. [ENOENT]
>
> *Directory2* or *directory1* is busy. [EBUSY]
>
> A component of either path prefix denies search permission. [EACCES]

Urmnt fails if one or more of the following is true:

> The effective user ID of the process is not super-user. [EPERM]
>
> *Directory1* is not a directory. [ENOTDIR]
>
> *Directory1* does not exist. [ENOENT]
>
> *Directory1* is busy. [EBUSY]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SUPPORT STATUS

Supported.

NAME

semctl − semaphore control operations

SYNOPSIS

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

int semctl (semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
     int val;
     struct semid_ds *buf;
     ushort *array;
} arg;
```

DESCRIPTION

Semctl provides a variety of semaphore control operations as specified by *cmd*.

Semctl executes the following *cmd*s with respect to the semaphore specified by *semid* and *semnum:*

GETVAL	Return the value of semval (see *intro*(2)). {READ}
SETVAL	Set the value of semval to *arg.val*. {ALTER} After successful execution of this *cmd*, *semctl* clears the semadj value corresponding to the specified semaphore in all processes.
GETPID	Return the value of sempid. {READ}
GETNCNT	Return the value of semncnt. {READ}
GETZCNT	Return the value of semzcnt. {READ}

The following *cmd*s return and set, respectively, every semval in the set of semaphores.

GETALL	Place semvals into array pointed to by *arg.array*. {READ}
SETALL	Set semvals according to the array pointed to by *arg.array*. {ALTER} After successful execution of this *cmd*, *semctl* clears the semadj values corresponding to each specified semaphore in all processes.

The following *cmd*s are also available:

IPC_STAT	Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in *intro*(2). {READ}
IPC_SET	Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*:

sem_perm.uid
sem_perm.gid
sem_perm.mode /* only low 9 bits */

This cmd can only be executed by a process that has an effective user ID equal to either that of super user or to the value of **sem_perm.uid** in the data structure associated with *semid*.

IPC_RMID — Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it.

This cmd can only be executed by a process that has an effective user ID equal to either that of super user or to the value of **sem_perm.uid** in the data structure associated with *semid*.

FAILURE CONDITIONS

Semctl fails if one or more of the following is true:

Semid is not a valid semaphore identifier. [EINVAL]

Semnum is less than zero or greater than **sem_nsems**. [EINVAL]

Cmd is not a valid command. [EINVAL]

Operation permission is denied to the calling process (see *intro*(2)). [EACCES]

Cmd is **SETVAL** or **SETALL** and the value to which semval is to be set is greater than the system imposed maximum. [ERANGE]

Cmd is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of super user and it is not equal to the value of **sem_perm.uid** in the data structure associated with *semid*. [EPERM]

Arg.buf points to an illegal address. [EFAULT]

RETURN VALUE

Upon successful completion, the return value depends on *cmd* as follows:

GETVAL	The value of semval.
GETPID	The value of sempid.
GETNCNT	The value of semncnt.
GETZCNT	The value of semzcnt.
All others	A value of 0.

Otherwise, the return value is −1 and *errno* indicates the error.

SEE ALSO

intro(2), semget(2), semop(2).

SUPPORT STATUS

Supported.

NAME

semget — get a set of semaphores

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;

DESCRIPTION

Semget returns the semaphore identifier associated with *key*.

Semget creates a semaphore identifier and its associated data structure and set containing *nsems* semaphores (see *intro*(2)) for *key* if one of the following is true:

Key is equal to **IPC_PRIVATE**.

Key does not already have a semaphore identifier associated with it, and (*semflg* & **IPC_CREAT**) is *true*.

After creation, *semget* initializes the data structure associated with the new semaphore identifier as follows:

Sets **sem_perm.cuid**, **sem_perm.uid**, **sem_perm.cgid**, and **sem_perm.gid** to the effective user ID and effective group ID, respectively, of the calling process.

Sets the low-order 9 bits of **sem_perm.mode** to the low-order 9 bits of *semflg*.

Sets **sem_nsems** to the value of *nsems*.

Sets **sem_otime** to 0 and **sem_ctime** to the current time.

FAILURE CONDITIONS

Semget fails if one or more of the following is true:

Nsems is either less than or equal to zero or greater than the system imposed limit. [EINVAL]

A semaphore identifier exists for *key* but operation permission (see *intro*(2)) as specified by the low-order 9 bits of *semflg* would not be granted. [EACCES]

A semaphore identifier exists for *key* but the number of semaphores in the set associated with it is less than *nsems*, and *nsems* is not equal to zero. [EINVAL]

A semaphore identifier does not exist for *key* and (*semflg* & **IPC_CREAT**) is *false*. [ENOENT]

The system imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded if the semaphore identifier was created as requested. [ENOSPC]

The system imposed limit on the maximum number of allowed semaphores system wide would be exceeded if the semaphore identifier was created as requested. [ENOSPC]

A semaphore identifier exists for *key* but ((*semflg* & IPC_CREAT) & (*semflg* & IPC_EXCL)) is *true*. [EEXIST]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the semaphore identifier
−1	error; **errno** indicates the error

SEE ALSO

intro(2), semctl(2), semop(2).

SUPPORT STATUS

Supported.

NAME

semop — semaphore operations

SYNOPSIS

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

int semop (semid, sops, nsops)
int semid;
struct sembuf (*sops)[];
int nsops;
```

DESCRIPTION

Semop automatically performs an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *Sops* is a pointer to the array of semaphore-operation structures. *Nsops* is the number of such structures in the array. Each structure includes the following members:

```
short   sem_num;   /* semaphore number */
short   sem_op;    /* semaphore operation */
short   sem_flg;   /* operation flags */
```

Semop performs each semaphore operation specified by *sem_op* on the corresponding semaphore specified by *semid* and *sem_num*.

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

Sem_op specifies one of three semaphore operations as follows:

If *sem_op* is a positive integer, add the value of *sem_op* to semval and, if (*sem_flg* & SEM_UNDO) is *true*, subtract the value of *sem_op* from the semadj value for the specified semaphore of the calling process. {ALTER}

If *sem_op* is a negative integer, one of the following occurs: {ALTER}

If semval (see *intro*(2)) is greater than or equal to the absolute value of *sem_op*, subtract the absolute value of *sem_op* from semval. Also, if (*sem_flg* & SEM_UNDO) is *true*, add the absolute value of *sem_op* to the semadj value (see *exit*(2)) for the specified semaphore of the calling process.

If semval is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is *true*, return immediately.

If semval is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is *false*, increment the semncnt associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

— Semval becomes greater than or equal to the absolute value of *sem_op*. When this occurs, decrement the value of semncnt associated with the specified semaphore, subtract the absolute value of *sem_op* from semval and, if (*sem_flg* & SEM_UNDO) is *true*, add the absolute value of *sem_op* is added to the semadj value for the specified semaphore of the calling process.

–

The semid for which the calling process is awaiting action is removed from the system (see *semctl*(2)). When this occurs, set *errno* to EIDRM, and return a value of −1.

–

The calling process receives a signal that is to be caught. When this occurs, decrement the value of semncnt associated with the specified semaphore, and resume exelcution of the calling process in the manner prescribed in *signal*(2).

If *sem_op* is zero, one of the following occurs: {READ}

If semval is zero, return immediately.

If semval is not equal to zero and (*sem_flg* & IPC_NOWAIT) is *true*, return immediately.

If semval is not equal to zero and (*sem_flg* & IPC_NOWAIT) is *false*, increment the semzcnt associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

–

Semval becomes zero. At that time, decrement the value of semzcnt associated with the specified semaphore.

–

The semid for which the calling process is awaiting action is removed from the system. When this occurs, set *errno* to EIDRM, and return a value of −1.

–

The calling process receives a signal that is to be caught. When this occurs, decrement the value of semzcnt associated with the specified semaphore, and resume execution of the calling process in the manner prescribed in *signal*(2).

FAILURE CONDITIONS

Semop fails if one or more of the following is true for any of the semaphore operations specified by *sops*:

Semid is not a valid semaphore identifier. [EINVAL]

Sem_num is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*. [EFBIG]

Nsops is greater than the system imposed maximum. [E2BIG]

Operation permission is denied to the calling process (see *intro*(2)). [EACCES]

The operation would result in suspension of the calling process but (*sem_flg* & IPC_NOWAIT) is *true*. [EAGAIN]

The limit on the number of individual processes requesting an SEM_UNDO would be exceeded. [ENOSPC]

The number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the limit. [EINVAL]

An operation would cause a semval to overflow the system imposed limit. [ERANGE]

An operation would cause a semadj value to overflow the system imposed limit. [ERANGE]

Sops points to an illegal address. [EFAULT]

RETURN VALUE

On error, *semop* returns −1.

If *errno* is EINTR, return is due to the receipt of a signal

If *errno* is EIDRM, return is due to the removal of *semid* from the system

Otherwise, *errno* indicates the error.

Upon successful completion, *semop* returns the value of semval at the time of the call for the last operation in the array pointed to by *sops*.

SEE ALSO

exec(2), exit(2), fork(2), intro(2), semctl(2), semget(2).

SUPPORT STATUS

Supported.

NAME

sernum — get serial number of current system

SYNOPSIS

int sernum ()

DESCRIPTION

Sernum retrieves hardware defined information identifying the current system and returns it as a positive number.

RETURN VALUE

A non-negative value is returned on successful completion or −1 is returned on error and **errno** indicates the error.

SUPPORT STATUS

Supported.

NAME

setpgrp — set process group ID

SYNOPSIS

int setpgrp ()

DESCRIPTION

Setpgrp sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

RETURN VALUE

Setpgrp returns the value of the new process group ID.

SEE ALSO

exec(2), fork(2), getpid(2), intro(2), kill(2), signal(2).

SUPPORT STATUS

Supported.

NAME

setuid, setgid — set user and group IDs

SYNOPSIS

int setuid (uid)
int uid;

int setgid (gid)
int gid;

DESCRIPTION

Setuid (*setgid*) sets the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is super-user, *setuid* sets the real user (group) ID and effective user (group) ID to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to *uid* (*gid*), *setuid* sets the effective user (group) ID to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but the saved set-user (group) ID from *exec*(2) is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

FAILURE CONDITIONS

Setuid (*setgid*) fails if the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not super-user. [EPERM]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

getuid(2), intro(2).

SUPPORT STATUS

Supported.

NAME

shmctl — shared memory control operations

SYNOPSIS

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

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```

DESCRIPTION

Shmctl provides a variety of shared memory control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT

Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro*(2). {READ}

IPC_SET

Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */
```

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of **shm_perm.uid** in the data structure associated with *shmid*.

IPC_RMID

Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and data structure associated with it.

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of **shm_perm.uid** in the data structure associated with *shmid*.

FAILURE CONDITIONS

Shmctl fails if one or more of the following is true:

Shmid is not a valid shared memory identifier. [EINVAL]

Cmd is not a valid command. [EINVAL]

Cmd is equal to IPC_STAT and {READ} operation permission is denied to the calling process (see *intro*(2)). [EACCES]

Cmd is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of super user and it is not equal to the value of **shm_perm.uid** in the data structure associated with *shmid*. [EPERM]

Buf points to an illegal address. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

intro(2), shmget(2), shmop(2).

SUPPORT STATUS

Supported.

NAME

shmget − get shared memory segment

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key, size, shmflg)
key_t key;
int size, shmflg;

DESCRIPTION

Shmget returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of size *size* bytes (see *intro*(2)) are created for *key* if one of the following is true:

Key is equal to **IPC_PRIVATE.**

Key does not already have a shared memory identifier associated with it, and (*shmflg* & **IPC_CREAT**) is *true*.

Upon creation, *shmget* initializes the data structure associated with the new shared memory identifier as follows:

Sets **shm_perm.cuid, shm_perm.uid, shm_perm.cgid,** and **shm_perm.gid** to the effective user ID and effective group ID, respectively, of the calling process.

Sets the low-order 9 bits of **shm_perm.mode** equal to the low-order 9 bits of *shmflg*, and sets **shm_segsz** to the value of *size*.

Sets **shm_lpid, shm_nattch, shm_atime,** and **shm_dtime** to 0.

Sets **shm_ctime** to the current time.

FAILURE CONDITIONS

Shmget fails if one or more of the following is true:

Size is less than the system imposed minimum or greater than the system imposed maximum. [EINVAL]

A shared memory identifier exists for *key*, but operation permission (see *intro*(2)) as specified by the low-order 9 bits of *shmflg* would not be granted. [EACCES]

A shared memory identifier exists for *key*, but the size of the segment associated with it is less than *size* and *size* is not equal to zero. [EINVAL]

A shared memory identifier does not exist for *key* and (*shmflg* & **IPC_CREAT**) is *false*. [ENOENT]

The system imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded if the shared memory identifier was created. [ENOSPC]

The amount of available physical memory is not sufficient to fill the request if the shared memory identifier and associated shared memory segment were created. [ENOMEM]

A shared memory identifier exists for *key* but ((*shmflg* & IPC_CREAT) & (*shmflg* & IPC_EXCL)) is *true*. [EEXIST]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer is the shared memory identifier
−1	error; **errno** indicates the error

SEE ALSO

intro(2), shmctl(2), shmop(2).

SUPPORT STATUS

Supported.

NAME

shmat, shmdt — shared memory operations

SYNOPSIS

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

char *shmat (shmid, shmaddr, shmflg)
int shmid;
char *shmaddr
int shmflg;

int shmdt (shmaddr)
char *shmaddr
```

DESCRIPTION

Shmat attaches the shared memory segment associated with the shared memory identifier *shmid* to the data segment of the calling process at the address specified by one of the following criteria:

- If *shmaddr* is equal to zero, attach the segment at the first available address as selected by the system.
- If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is true, attach the segment at the address given by (*shmaddr* — (*shmaddr* modulus SHMLBA)).
- If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is false, attach the segment at the address given by *shmaddr*.

Shmat attaches the segment for reading if (*shmflg* & SHM_RDONLY) is true {READ}; otherwise, *shmat* attaches the segment for reading and writing {READ/WRITE}.

Shmdt detaches from data segment of the calling process the shared memory segment located at the address specified by *shmaddr*.

FAILURE CONDITIONS

Shmat fails and does not attach the shared memory segment if one or more of the following is true:

- *Shmid* is not a valid shared memory identifier. [EINVAL]
- Operation permission is denied to the calling process (see *intro*(2)). [EACCES]
- The available data space is not large enough to accommodate the shared memory segment. [ENOMEM]
- *Shmaddr* is not equal to zero, and (*shmaddr* — (*shmaddr* modulus SHMLBA)) is an invalid address. [EINVAL]
- *Shmaddr* is not equal to zero, (*shmflg* & SHM_RND) is false, and the value of *shmaddr* is an invalid address. [EINVAL]
- The number of shared memory segments attached to the calling process would exceed the system imposed limit. [EMFILE]
- *Shmdt* detaches from the data segment of the calling process the shared memory segment located at the address specified by *shmaddr*. [EINVAL]

Shmdt fails and does not detach the shared memory segment if *shmaddr* is not the data segment start address of a shared memory segment. [EINVAL]

RETURN VALUE

On successful completion, *shmat* returns the data segment start address of the attached shared memory segment and *shmdt* returns the value of 0. On unsuccessful completion, a value of −1 is returned; **errno** indicates the error.

SEE ALSO

exec(2), exit(2), fork(2), intro(2), shmctl(2), shmget(2).

SUPPORT STATUS

Supported.

NAME

signal — specify what to do upon receipt of a signal

SYNOPSIS

#include <signal.h>

int (*(signal (sig, func))()
int sig;
void (*(func)();

DESCRIPTION

Signal allows the calling process to choose one of three ways to handle the receipt of a specific signal. *Sig* specifies the signal and *func* specifies the choice.

A call to *signal* cancels a pending signal *sig* except for a pending SIGKILL signal.

ARGUMENTS

Sig can be assigned any one of the following except SIGKILL:

SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03	quit
SIGILL	04	invalid instruction (not reset when caught)
SIGTRAP	05	trace trap (not reset when caught)
SIGIOT	06	IOT instruction
SIGEMT	07	EMT instruction
SIGFPE	08	floating point exception
SIGKILL	09	kill (cannot be caught or ignored)
SIGBUS	10	bus error
SIGSEGV	11	segmentation violation
SIGSYS	12	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user-defined signal 1
SIGUSR2	17	user-defined signal 2
SIGCLD	18	death of a child (see *WARNING* below)
SIGPWR	19	power fail (see *WARNING below)*

Func is assigned one of three values: SIG_DFL, SIG_IGN, or a *function address*. The actions specified by these values are:

SIG_DFL — terminate process upon receipt of signal

Upon receipt of the signal *sig*, terminate the receiving process with all of the consequences outlined in *exit*(2) and make a *core image* in the current working directory of the receiving process if the following conditions are met:

- The effective user ID and the real user ID of the receiving process are equal.
- An ordinary file named core exists and is writable or can be created. If the file must be created, *signal* assigns the following properties:

- mode of 0666 modified by the file creation mask (see *umask*(2))
- file owner ID that is the same as the effective user ID of the receiving process
- file group ID that is the same as the effective group ID of the receiving process

- *Sig* is one of the following:

SIGBUS	SIGFPE
SIGEMT	SIGILL
SIGIOT	SIGQUIT
SIGSEGV	SIGSYS
SIGTRAP	

SIG_IGN – ignore signal

Ignore the signal *sig*. The signal **SIGKILL** cannot be ignored.

function address – catch signal

Upon receipt of the signal *sig*, the receiving process executes the signal-catching function pointed to by *func*. *Signal* passes the signal number *sig* as the only argument to the signal-catching function. Before entering the signal-catching function, *signal* sets the value of *func* for the caught signal to **SIG_DFL** unless the signal is **SIGILL**, **SIGTRAP**, or **SIGPWR**.

Upon return from the signal-catching function, the receiving process resumes execution at the point it was interrupted.

When a signal that is to be caught occurs

- during a *read*, a *write*, an *open*, or an *ioctl* system call on a slow device (like a terminal; but not a file),
- during a *pause* system call, or
- during a *wait* system call that does not return immediately due to the existence of a previously stopped or zombie process,

signal executes the signal catching function and then the interrupted system call returns a −1 to the calling process with *errno* set to EINTR.

Note: The signal **SIGKILL** cannot be caught.

FAILURE CONDITIONS

Signal fails if one or more of the following is true:

- *Sig* is an invalid signal number, including **SIGKILL**. [EINVAL]
- *Func* points to an invalid address. [EFAULT]

RETURN VALUE

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

kill(2), pause(2), ptrace(2), wait(2), setjmp(3C), kill(1).

WARNING

Two other signals that behave differently than the signals described above exist in this release of the system; they are:

SIGCLD	18	death of a child (reset when caught)
SIGPWR	19	power fail (not reset when caught)

There is no guarantee that, in future releases of the UNIX System, these signals will continue to behave as described below; they are included only for compatibility with other versions of the UNIX System. Their use in new programs is strongly discouraged.

For these signals, *func* is assigned one of three values: SIG_DFL, SIG_IGN, or a *function address*. The actions prescribed by these values of are as follows:

SIG_DFL – ignore signal
    Ignore the signal.

SIG_IGN – ignore signal
    Ignore the signal. Also, if *sig* is SIGCLD, the child processes of the calling process do not create zombie processes when they terminate; see *exit*(2).

function address – catch signal
    If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is SIGCLD except that while the process is executing the signal-catching function any received SIGCLD signals are queued and the signal-catching function is continually reentered until the queue is empty.

The SIGCLD affects two other system calls (*wait*(2), and *exit*(2)) in the following ways:

wait If the *func* value of SIGCLD is set to SIG_IGN and a *wait* is executed, the *wait* blocks until all of the calling process's child processes terminate; it then returns a value of −1 with *errno* set to ECHILD.

exit If in the parent process of the exiting process the *func* value of SIGCLD is set to SIG_IGN, the exiting process does not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.

SUPPORT STATUS

Supported.

NAME

slink — link files across file systems

SYNOPSIS

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

DESCRIPTION

Path1 points to the pathname of an existing file. *Path2* points to the pathname of the new directory entry to be created. *Slink* creates a new link (directory entry) for the existing file.

Slink is very similar to the *link* command except the inode link count for *path1* is not incremented.

FAILURE CONDITIONS

Slink fails and does not create a link if one or more of the following is true:

A component of either path prefix is not a directory. [ENOTDIR]

A component of either path prefix does not exist. [ENOENT]

A component of either path prefix denies search permission. [EACCES]

The file named by *path1* does not exist. [ENOENT]

The link named by *path2* exists. [EEXIST]

The file named by *path1* is a directory and the effective user ID is not super-user. [EPERM]

Path2 points to a null path name. [ENOENT]

The requested link requires writing in a directory with a mode that denies write permission. [EACCES]

The requested link requires writing in a directory on a read-only file system. [EROFS]

Path points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

unlink(2), inode(4).

SUPPORT STATUS

Supported.

NAME
stat, fstat – get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>

int stat (path, buf)
char *path;
struct stat *buf;

int fstat (fildes, buf)
int fildes;
struct stat *buf;

DESCRIPTION
Stat obtains information about the named file specified by *path*. *Path* points to a path name naming a file.

Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

Similarly, *fstat* obtains information about an open file known by the file descriptor *fildes*, obtained from a successful *open*, *creat*, *dup*, *fcntl*, or *pipe* system call.

ARGUMENTS
Buf is a pointer to a *stat* structure into which information is placed concerning the file.

The contents of the structure pointed to by *buf* include the following members:

```
ushort st_mode;     /* File mode; see mknod(2) */
ino_t  st_ino;      /* Inode number */
dev_t  st_dev;      /* ID of device containing */
                    /* a directory entry for this file */
dev_t  st_rdev;     /* ID of device; this entry is */
                    /* defined only for character */
                    /* special or block special files */
short  st_nlink;    /* Number of links */
ushort st_uid;      /* User ID of the file owner */
ushort st_gid;      /* Group ID of the file group */
off_t  st_size;     /* File size in bytes */
time_t st_atime;    /* Time of last access */
time_t st_mtime;    /* Time of last data modification */
time_t st_ctime;    /* Time of last file status change */
                    /* Times measured in seconds since */
                    /* 00:00:00 GMT, Jan. 1, 1970 */
```

Three of these structure members are altered as follows:

st_atime (Time when file data was last accessed) Changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *read*(2).

st_mtime (Time when data was last modified) Changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *write*(2).

st_ctime (Time when file status was last changed) Changed by the following system calls: *chmod*(2), *chown*(2), *creat*(2), *link*(2), *mknod*(2), *pipe*(2), *unlink*(2), *utime*(2), and *write*(2).

FAILURE CONDITIONS

Stat fails if one or more of the following is true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Buf or *path* points to an invalid address. [EFAULT]

Fstat fails if one or more of the following is true:

Fildes is not a valid open file descriptor. [EBADF]

Buf points to an invalid address. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

chmod(2), chown(2), creat(2), link(2), mknod(2), pipe(2), read(2), time(2), unlink(2), utime(2), write(2).

SUPPORT STATUS

Supported.

NAME

stime − set time

SYNOPSIS

int stime (tp)
long *tp;

DESCRIPTION

Stime sets the system time and date. *Tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

FAILURE CONDITIONS

Stime fails if the effective user ID of the calling process is not super-user. [EPERM]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

time(2).

SUPPORT STATUS

Supported.

NAME

swrite – synchronously write on a file

SYNOPSIS

int swrite (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION

Swrite attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*. Control is not returned to the program until the block has been actually written to the file device and the inode associated with the file has been updated and if necessary written to the file device. If the file is not associated with a block device, the behavior of *swrite* is the same as *write*. In all other respects the *swrite* is the same as *write*.

RETURN VALUE

non-negative	successful completion; the non-negative integer indicates the actual number of bytes written
−1	error; **errno** indicates the error

SEE ALSO

write(2), open(2).

SUPPORT STATUS

Supported.

NAME

sync — update super-block

SYNOPSIS

void sync ( )

DESCRIPTION

Sync writes out all information in memory that should be on disk; this includes modified super blocks, modified i-nodes, and delayed block I/O.

Sync should be used by programs which examine a file system, such as *fsck*, *df*, etc. *Sync* is mandatory before a boot.

Sync schedules the write, but the writing may not actually be complete when *sync* returns.

SUPPORT STATUS

Supported.

NAME
time − get time

SYNOPSIS
long time ((long *) 0)

long time (tloc)
long *tloc;

DESCRIPTION
Time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If *tloc* (taken as an integer) is non-zero, *time* also stores the return value in the location to which *tloc* points.

FAILURE CONDITIONS
Time fails if *tloc* points to an illegal address. [EFAULT]

RETURN VALUE
Upon successful completion, *time* returns the value of time. Otherwise, *time* returns a value of −1 and sets *errno* to indicate the error.

SEE ALSO
stime(2).

SUPPORT STATUS
Supported.

NAME

times — get process and child process times

SYNOPSIS

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

long times (buffer)
struct tms *buffer;

DESCRIPTION

Times fills the structure pointed to by *buffer* with time-accounting information. The following is this contents of the structure:

```
struct  tms {
        time_t  tms_utime;
        time_t  tms_stime;
        time_t  tms_cutime;
        time_t  tms_cstime;
};
```

Tms_utime the CPU time used while executing instructions in the user space of the calling process.

Tms_stime the CPU time used by the system on behalf of the calling process.

Tms_cutime the sum of the *tms_utime*s and *tms_cutime*s of the child processes.

Tms_cstime the sum of the *tms_stime*s and *tms_cstime*s of the child processes.

This information comes from the calling process and each of its terminated child processes for which it has executed a *wait*. All times are in 60ths of a second.

FAILURE CONDITIONS

Times fails if *buffer* points to an illegal address. [EFAULT]

RETURN VALUE

Upon successful completion, *times* returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of *times* to another.

If *times* fails, it returns a −1 and sets *errno* to indicate the error.

SEE ALSO

exec(2), fork(2), time(2), wait(2).

SUPPORT STATUS

Supported.

NAME

tslice – set/get time slice

SYNOPSIS

int tslice (ts)
unsigned int ts;

DESCRIPTION

Tslice sets the system-wide timeslice. *Ts* is the new timeslice value in hundreths of seconds. If *ts* is zero then the current timeslice value is returned. Values of *ts* greater than 5461 (approximately 54 seconds) are reduced to 5461.

FAILURE CONDITIONS

Tslice fails if the effective user ID of the calling process is not superuser and *ts* is not zero.

RETURN VALUE

−1 error; **errno** indicates the error otherwise the new setting is returned.

RESTRICTIONS

Timeslice values less than 100 ms or greater than 3 seconds are not recommended.

SUPPORT STATUS

Supported.

NAME

ulimit – get and set user limits

SYNOPSIS

long ulimit (cmd, newlimit)
int cmd;
long newlimit;

DESCRIPTION

This function controls process limits. The *cmd* values available are:

1 Get the file size limit for the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2 Set the file size limit for the process to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit.

3 Get the maximum possible break value. See *brk*(2).

FAILURE CONDITIONS

Ulimit fails and the limit is unchanged if a process with an effective user ID other than super-user attempts to increase its file size limit. [EPERM]

RETURN VALUE

non-negative integer	successful completion
−1	error; errno indicates the error

SEE ALSO

brk(2), write(2).

SUPPORT STATUS

Supported.

NAME

umask — set and get file creation mask

SYNOPSIS

int umask (cmask)
int cmask;

DESCRIPTION

Umask sets the file mode creation mask of the process to *cmask* and returns the previous value of the mask. Only the low-order 9 bits of *cmask* and the file mode creation mask are used.

RETURN VALUE

Umask returns the previous value of the file mode creation mask.

SEE ALSO

chmod(2), creat(2), mknod(2), open(2), mkdir(1), sh(1).

SUPPORT STATUS

Supported.

NAME

umount — unmount a file system

SYNOPSIS

int umount (spec)
char *spec;

DESCRIPTION

Umount unmounts a previously mounted file system contained on the block special device identified by *spec*. *Spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

Only the super-user may invoke *umount*.

FAILURE CONDITIONS

Umount fails if one or more of the following is true:

The effective user ID of the process is not super-user. [EPERM]

Spec does not exist. [ENXIO]

Spec is not a block special device. [ENOTBLK]

Spec is not mounted. [EINVAL]

A file on *spec* is busy. [EBUSY]

Spec points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

mount(2).

SUPPORT STATUS

Supported.

NAME

uname — get name of current UNIX system

SYNOPSIS

#include <sys/utsname.h>

int uname (name)
struct utsname *name;

DESCRIPTION

Uname stores information identifying the current UNIX system in the structure pointed to by *name*.

Uname returns a null-terminated character string naming the current UNIX system.

Uname uses the structure defined in <sys/utsname.h> whose members are:

```
char    sysname[9];   /* system name */
char    nodename[9];
                      /* name the system is known by on a
                         communications network */
char    release[9];   /* release number */
char    version[9];   /* version number */
char    machine[9];   /* name of hardware that UNIX is
                         running on */
```

FAILURE CONDITIONS

Uname fails if *name* points to an invalid address. [EFAULT]

RETURN VALUE

non-negative integer	successful completion
−1	error; **errno** indicates the error

SEE ALSO

uname(1).

SUPPORT STATUS

Supported.

NAME

unlink – remove directory entry

SYNOPSIS

int unlink (path)
char *path;

DESCRIPTION

Unlink removes the directory entry named by the path name pointed to be *path*.

When all links to a file have been removed and no process has the file open, the system frees the space occupied by the file and the file ceases to exist. If one or more processes have the file open when the last link is removed, the system postpones the removal until all references to the file are closed.

FAILURE CONDITIONS

Unlink fails if one or more of the following is true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Write permission is denied on the directory containing the link to be removed. [EACCES]

The named file is a directory and the effective user ID of the process is not super-user. [EPERM]

The entry to be unlinked is the mount point for a mounted file system. [EBUSY]

The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed. [ETXTBSY]

The directory entry to be unlinked is part of a read-only file system. [EROFS]

Path points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; **errno** indicates the error

SEE ALSO

close(2), link(2), open(2), rm(1).

SUPPORT STATUS

Supported.

NAME

ustat — get file system statistics

SYNOPSIS

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

int ustat (dev, buf)
int dev;
struct ustat *buf;
```

DESCRIPTION

Ustat returns information about a mounted file system. *Dev* is a device number identifying a device containing a mounted file system. *Buf* is a pointer to a *ustat* structure that includes to following elements:

```
daddr_t f_tfree;        /* Total free blocks */
ino_t   f_tinode;       /* Number of free inodes */
char    f_fname[6];     /* Filsys name */
char    f_fpack[6];     /* Filsys pack name */
```

FAILURE CONDITIONS

Ustat fails if one or more of the following is true:

Dev is not the device number of a device containing a mounted file system. [EINVAL]

Buf points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0 successful completion
−1 error; errno indicates the error

SEE ALSO

stat(2), fs(4).

SUPPORT STATUS

Supported.

NAME

utime — set file access and modification times

SYNOPSIS

#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;

DESCRIPTION

Utime sets the access and modification times of the file specified by *path*. *Path* points to a path name naming a file.

If *times* is NULL, *utime* sets the access and modification times of the file to the current time. A process must be the owner of the file or have write permission to use *utime* in this manner.

If *times* is not NULL, *utime* interprets *times* as a pointer to a *utimbuf* structure and sets the access and modification times to the values contained in the designated structure. Only the owner of the file or the super-user may use *utime* this way.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct  utimbuf {
        time_t  actime;      /* access time */
        time_t  modtime;     /* modification time */
};
```

FAILURE CONDITIONS

Utime fails if one or more of the following is true:

The named file does not exist. [ENOENT]

A component of the path prefix is not a directory. [ENOTDIR]

Search permission is denied by a component of the path prefix. [EACCES]

The effective user ID is not super-user and not the owner of the file and *times* is not NULL. [EPERM]

The effective user ID is not super-user and not the owner of the file and *times* is NULL and write access is denied. [EACCES]

The file system containing the file is mounted read-only. [EROFS]

Times is not NULL and points outside the allocated address space of the process. [EFAULT]

Path points outside the allocated address space of the process. [EFAULT]

RETURN VALUE

0	successful completion
−1	error; **errno** indicates the error

SEE ALSO

stat(2).

SUPPORT STATUS

Supported.

NAME

wait — wait for child process to stop or terminate

SYNOPSIS

int wait (stat_loc)
int *stat_loc;

int wait ((int *)0)

DESCRIPTION

Wait suspends the calling process until it receives a signal that is to be caught (see *signal*(2)), or until any one of the child processes of the calling process stops in a trace mode (see *ptrace*(2)) or terminates. If a child process stopped or terminated prior to the *wait* call, *wait* returns immediately.

If *stat_loc* (taken as an integer) is non-zero, 16 bits of *status* information are stored in the low order 16 bits of the location pointed to by *stat_loc*. *Status* differentiates between stopped and terminated child processes and if the child process terminated, status identifies the cause of termination and passes useful information to the parent:

> If the child process stopped, the high order 8 bits of status contain the number of the signal that caused the process to stop and the low order 8 bits are equal to 0177.
>
> If the child process terminated due to an *exit* call, the low order 8 bits of status are zero and the high order 8 bits contain the low order 8 bits of the argument that the child process passed to *exit*; see *exit*(2).
>
> If the child process terminated due to a signal, the high order 8 bits of status are zero and the low order 8 bits contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 200) is set, *wait* produced a *core image*; see *signal*(2).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes; see *intro*(2).

FAILURE CONDITIONS

Wait fails and returns immediately if one or more of the following is true:

> The calling process has no existing unwaited-for child processes. [ECHILD]
>
> *Stat_loc* points to an illegal address. [EFAULT]

RETURN VALUE

If *wait* returns due to the receipt of a signal, a value of −1 is returned to the calling process and *errno* is set to EINTR.

If *wait* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process.

Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO

exec(2), exit(2), fork(2), intro(2), pause(2), ptrace(2), signal(2).

WARNING

See *WARNING* in *signal*(2).

SUPPORT STATUS

Supported.

NAME

write – write on a file

SYNOPSIS

int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION

Write attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*. *Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return, *write* increments the file pointer by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, *write* sets the file pointer to the end of the file prior to each write.

If the number of bytes to be written exceeds a defined limit (e.g., the *ulimit* (see *ulimit*(2)) or a physical limit (e.g., the physical end of a medium), *write* writes as many bytes as possible without exceeding the limit.

For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes returns 20. The next write of a non-zero number of bytes returns a failure.

If the file being written is a pipe (or FIFO), no partial writes are permitted. Thus, the write fails if a write of *nbyte* bytes would exceed a limit.

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, then the writes to a full pipe (or FIFO) return a count of 0. Otherwise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

A write to a regular file is blocked if a region of the file is locked by another process using *lockf*(2) or *lockf*(3X) (or *fcntl*(2)) with mandatory file/record locking enabled on the file (see *chmod*(2)). In these cases, if *lockf*(2) was used to lock the file region or if *lockf* was uused with the O_NDELAY flag clear, the write sleeps until the blocking record lock is removed.

FAILURE CONDITIONS

Write fails and does not change the file pointer if one or more of the following is true:

Fildes is not a valid file descriptor open for writing. [EBADF]

An attempt is made to write to a pipe that is not open for reading by any process. [EPIPE and SIGPIPE signal]

An attempt is made to write a file that exceeds the file size limit of the process or the maximum file size. See *ulimit*(2). [EFBIG]

Buf points outside the allocated address space of the process. [EFAULT]

A signal was caught during the *write* system call. [EINTR]

Mandatory file/record locking is in effect, O_NDELAY was set, and there was a blocking record lock. [EAGAIN]

The total amount of system memory available when reading via raw I/O is temporarily insufficent. [EAGAIN]

The system record lock table was full, so the write could not go to sleep until the blocking record lock was removed. [ENOLCK]

The write was going to go to sleep and cause a deadlock situation due to previous mandatory locks on the file. [EDEADLK]

RETURN VALUE

non-negative integer	successful completion; the non-negative integer indicates the actual number of bytes written
−1	error; *errno* indicates the error

SEE ALSO

creat(2), dup(2), fcntl(2), lockf(3X), lockf(2), lseek(2), open(2), pipe(2), ulimit(2).

SUPPORT STATUS

Supported.

NAME

intro — introduction to subroutines and libraries

SYNOPSIS

#include <stdio.h>

#include <math.h>

DESCRIPTION

This section describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Certain major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute the Standard C Library *libc*, which is automatically loaded by the C compiler, *cc*(1). The link editor *ld*(1) searches this library under the **−lc** option. Declarations for some of these functions may be obtained from **#include** files indicated on the appropriate pages.

(3F) These functions constitute the FORTRAN intrinsic function library, *libF77*. These functions are automatically available to the FORTRAN programmer and require no special invocation of the compiler.

(3M) These functions constitute the Math Library, *libm*. They are automatically loaded as needed by the FORTRAN compiler *f77*(1). They are not automatically loaded by the C compiler, *cc*(1); however, the link editor searches this library under the **−lm** option. Declarations for these functions may be obtained from the **#include** file **<math.h>**.

(3S) These functions constitute the *standard I/O package*. This package is described extensively in *stdio*(3S). These functions are in the library *libc*, already mentioned. Declarations for these functions may be obtained from the **#include** file **<stdio.h>**.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

DEFINITIONS

Character
  any bit pattern which fits into a byte on the machine

Null character
  a character with value 0, represented in the C language as '\0'

Character array
  a sequence of characters

Null-terminated character array
  a sequence of characters, the last of which is the *null character*

String
  a null-terminated character array

Null string
  is a character array containing only the null character.

NULL pointer

the value that is obtained by casting **0** into a pointer. The C language guarantees that this does not match that of any valid pointer, so many functions return the *NULL* pointer to indicate an error. *NULL* is defined as **0** in **<stdio.h>**; if **<stdio.h>** is not being used, *NULL* can be user defined.

Generic function

a FORTRAN intrinsic functions which returns a value that is the same type as the argument

FILES

/lib/libc.a
/usr/lib/libF77.a
/lib/libm.a

SEE ALSO

intro(2), stdio(3S), math(5), ar(1), cc(1), f77(1), ld(1), lint(1), nm(1).

DIAGNOSTICS

Functions in the C and Math Library (3C and 3M) may return the conventional values **0** or **HUGE** (the largest single-precision floating-point number) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* (see *intro*(2)) is set to the value EDOM or ERANGE.

Because many of the FORTRAN intrinsic functions use the routines found in the Math Library, the same conventions apply.

WARNING

Many of the functions in the libraries call and/or refer to other functions and external variables described in this section and in section 2 (*System Calls*). If a program inadvertantly defines a function or external variable with the same name, the presumed library version of the function or external varible may not be loaded. the *lint*(1) program checker reports name conflicts of this kind a "multiple declarations" of the names in question. Definitions for sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the **−l** option (for example, **−l***m* includes definitions for the Math Library, section 3M). Use of *lint* is highly recommended.

SUPPORT STATUS

Supported.

NAME

a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS

long a64l (s)
char *s;

char *l64a (l)
long l;

DESCRIPTION

These functions are used to maintain numbers stored in *base-64* ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a *digit* in a radix-64 notation.

The characters used to represent *digits* are:

characters	digits represented
.	0
/	1
0-9	2–11
A-Z	12–37
a-z	38–63

A64l takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. If the string pointed to by *s* contains more than six characters, *a64l* uses the first six.

L64a takes a **long** argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, *l64a* returns a pointer to a null string.

RESTRICTIONS

The value returned by *l64a* is a pointer into a static buffer, the contents of which are overwritten by each call.

SUPPORT STATUS

Supported.

NAME

abort — generate an IOT fault

SYNOPSIS

int abort ( )

DESCRIPTION

Abort sends an IOT signal to the process, which then terminates with a core dump, unless the signal is caught or ignored.

If SIGIOT is caught or ignored, *abort* returns control to the process. The value returned is that of the *kill*(2) system call.

SEE ALSO

exit(2), kill(2), signal(2), adb(1), sdb(1).

DIAGNOSTICS

If SIGIOT is neither caught nor ignored, and the current directory is writable, the process produces a core dump and the shell writes

abort — core dumped.

SUPPORT STATUS

Supported.

NAME

abort — terminate Fortran program

SYNOPSIS

call abort ( )

DESCRIPTION

Abort terminates the program which calls it, and closes all open files, truncating them at the current position of the file pointer.

DIAGNOSTICS

When invoked, *abort* prints

Fortran abort routine called

on the standard error output.

SEE ALSO

abort(3C).

SUPPORT STATUS

Not supported.

NAME

abs — return integer absolute value

SYNOPSIS

int abs (i)
int i;

DESCRIPTION

Abs returns the absolute value of its integer operand.

RESTRICTIONS

In two's-complement representation, the absolute value of the negative integer with largest magnitude is undefined; this error is ignored.

SEE ALSO

floor(3M).

SUPPORT STATUS

Supported.

NAME

abs, iabs, dabs, cabs, zabs — Fortran absolute value

SYNOPSIS

```
integer i1, i2
real r1, r2
double precision dp1, dp2
complex cx1, cx2
double complex dx1, dx2

r2 = abs(r1)

i2 = iabs(i1)
i2 = abs(i1)

dp2 = dabs(dp1)
dp2 = abs(dp1)

cx2 = cabs(cx1)
cx2 = abs(cx1)

dx2 = zabs(dx1)
dx2 = abs(dx1)
```

DESCRIPTION

Abs is the absolute value functions.

Iabs returns the integer absolute value of its integer argument.

Dabs returns the double-precision absolute value of its double-precision argument.

Cabs returns the complex absolute value of its complex argument.

Zabs returns the double-complex absolute value of its double-complex argument.

The generic form, *abs*, returns the absolute value of its argument; the returned value is the same type as the argument.

SEE ALSO

floor(3M).

SUPPORT STATUS

Not supported.

NAME

aimag, dimag — Fortran imaginary part of complex argument

SYNOPSIS

real r
complex cxr
double precision dp
double complex cxd

r = aimag(cxr)

dp = dimag(cxd)

DESCRIPTION

Aimag returns the imaginary part of its single-precision complex argument.

Dimag returns the double-precision imaginary part of its *double-complex* argument.

SUPPORT STATUS

Not supported.

NAME

aint, dint — Fortran integer part intrinsic function

SYNOPSIS

real r1, r2
double precision dp1, dp2

r2 = aint(r1)

dp2 = dint(dp1)
dp2 = aint(dp1)

DESCRIPTION

Aint returns the truncated value of its real argument as a real.

Dint returns the truncated value of its double-precision argument as a double-precision value.

The generic form *aint* returns the truncated value of its argument; the returned value is the same type as the argument.

SUPPORT STATUS

Not supported.

NAME

assert — verify program assertion

SYNOPSIS

#include <assert.h>

assert (expression)
int expression;

DESCRIPTION

Assert is useful for putting diagnostics into programs. When executed, if *expression* is false (zero), *assert* prints

Assertion failed: *expression*, file *xyz*, line *nnn*

on the standard error output and aborts. In the error message, *xyz* is the name of the source file; *nnn* is the source line number of the *assert* statement.

Compiling with the preprocessor option **−DNDEBUG** (see *cpp* (1)), or with the preprocessor control statement *#define NDEBUG* ahead of the *#include <assert.h>* statement, prevents compilation of assertions, effectively removing them from the program.

SEE ALSO

abort(3C), cpp(1).

SUPPORT STATUS

Supported.

NAME

j0, j1, jn, y0, y1, yn — Bessel functions

SYNOPSIS

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
int n;
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

DESCRIPTION

J0 and *j1* return Bessel functions of *x* of the first kind, of orders 0 and 1 respectively. *Jn* returns the Bessel function of *x* of the first kind of order *n*.

Y0 and *y1* return the Bessel functions of *x* of the second kind, of orders 0 and 1 respectively. *Yn* returns the Bessel function of *x* of the second kind of order *n*. The value of *x* must be positive.

DIAGNOSTICS

When non-positive arguments are specified, *y0*, *y1* and *yn*

— return the value negative **HUGE**
— set *errno* to **EDOM**
— print a message indicating DOMAIN error on the standard error output

When arguments too large in magnitude are specified, *j0*, *j1* and *jn*

— return the value 0
— set *errno* to **ERANGE**
— print a message indicating TLOSS error on the standard error output

The process continues.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

SUPPORT STATUS

Supported.

NAME

and, or, xor, not, lshift, rshift — Fortran bitwise boolean functions

SYNOPSIS

integer i, j, k
real a, b, c
double precision dp1, dp2, dp3

k = and(i, j)
c = or(a, b)
j = xor(i, a)
j = not(i)
k = lshift(i, j)
k = rshift(i, j)

DESCRIPTION

The generic intrinsic boolean functions *and*, *or* and *xor* return the value of the binary operations on their arguments.

Not is a unary operator returning the one's complement of its argument.

Lshift and *rshift* return the value of *i* shifted left or right, respectively, the number of times specified by *j*.

The boolean functions are defined for all data types as arguments and return values. Where required, the compiler generates appropriate type conversions.

NOTE

Although defined for all data types, use of a boolean function on any but integer data gives an unpredictable result.

WARNINGS

The shift functions may deliver unexpected results when passed large shift values.

SUPPORT STATUS

Not supported.

NAME

bsearch – binary search

SYNOPSIS

char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)
unsigned nel;
int (*compar)( );

DESCRIPTION

Bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function.

ARGUMENTS

Key points to the datum to be sought in the table.

Base points to the element at the base of the table.

Nel is the number of elements in the table.

Compar is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according to whether the first argument is to be considered less than, equal to, or greater than the second.

DIAGNOSTICS

Bsearch returns a NULL pointer if the key cannot be found in the table.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO

lsearch(3C), hsearch(3C), qsort(3C), tsearch(3C).

SUPPORT STATUS

Supported.

NAME

clock – report CPU time used

SYNOPSIS

long clock ( )

DESCRIPTION

Clock returns the amount of CPU time (in microseconds) used since the first call to *clock*. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed *wait*(2) or *system*(3S).

The resolution of the clock is 16.667 milliseconds.

SEE ALSO

times(2), wait(2), system(3S).

RESTRICTIONS

The value returned by *clock* is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned wraps around after accumulating only 2147 seconds of CPU time (about 36 minutes). Therefore, for extremely long running processes, *clock* is inaccurate.

SUPPORT STATUS

Supported.

NAME

conjg, dconjg — Fortran complex conjugate intrinsic function

SYNOPSIS

complex cx1, cx2
double complex dx1, dx2

cx2 = conjg(cx1)

dx2 = dconjg(dx1)

DESCRIPTION

Conjg returns the complex conjugate of its complex argument.

Dconjg returns the double-complex conjugate of its double-complex argument.

SUPPORT STATUS

Not supported.

NAME

toupper, tolower, _toupper, _tolower, toascii — translate characters

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

Tolower and *toupper* translate characters from one case to another. They accept arguments in the ranges of *getc*(3S) (the integers from −1 through 255), and return unchanged those characters which are not to be translated.

Toupper returns the upper-case character corresponding to its lower-case argument.

Tolower returns the lower-case character corresponding to its upper-case argument.

_toupper and *_tolower* are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *_toupper* requires a lower-case letter as its argument, and returns the corresponding upper-case letter. *_tolower* requires an upper-case letter as its argument, and returns the corresponding lower-case letter. Arguments outside the domain yield undefined results.

Toascii returns its argument with all bits turned off that are not part of a standard ASCII character for compatibility with other systems.

SEE ALSO

ctype(3C), getc(3S).

SUPPORT STATUS

Supported.

NAME

crypt, setkey, encrypt − generate DES encryption

SYNOPSIS

char *crypt (key, salt)
char *key, *salt;

void setkey (key)
char *key;

void encrypt (block, edflag)
char *block;
int edflag;

DESCRIPTION

Crypt

Crypt is the password encryption function. It is based on the NBS Data Encryption Standard (DES), with variations which (among other things) make it impossible for someone to implement in hardware a scheme to decode encrypted information, a typical means of breaking encryption.

Key is a password typed by the user. *Salt* is a two-character string chosen from the set [a-zA-Z0-9./]; this string is used to complicate the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.

Setkey

Setkey sets the key which is used with the above mentioned algorithm to encrypt or decrypt the string *block* with the function *encrypt*.

Key is a character array of length 64 containing only the characters with numerical value 0 and 1. *Setkey* divides this string into groups of 8, ignores the low-order bit in each group; *setkey* then sets this 56-bit key into the machine.

Encrypt

Encrypt encrypts or decrypts *block*, according to the value of *edflag*. If *edflag* is zero, the argument is encrypted; if non-zero, it is decrypted.

Block is a character array of length 64 containing only the characters with numerical value 0 and 1. *Encrypt* modifies the argument array in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*.

SEE ALSO

pgetpass(3C), passwd(4), login(1), passwd(1).

RESTRICTIONS

The return value points to static data that are overwritten by each call.

Outside the United States, *encrypt* does not perform the decrypt function. If the *edflag* is non-zero, *encrypt* exits with a −1 status.

SUPPORT STATUS

Supported. *Setkey* is available only in the United States.

NAME

ctermid − generate file name for terminal

SYNOPSIS

#include <stdio.h>

char *ctermid(s)
char *s;

DESCRIPTION

Ctermid generates the path name of the controlling terminal for the current process, and stores it in a string.

If *s* is a NULL pointer, *ctermid* stores the string in an internal static area and returns the address of that area. The next call to *ctermid* overwrites the contents of that static area.

If *s* is not a NULL pointer, *ctermid* assumes that *s* points to a character array of at least **L_ctermid** elements. *Cterm* places the path name in this array and returns the value of *s*. The constant **L_ctermid** is defined in the *<stdio.h>* header file.

NOTES

The difference between *ctermid* and *ttyname*(3C) is that *ttyname* must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while *ctermid* returns a string (/dev/tty) that refers to the terminal if used as a file name. Thus *ttyname* is useful only if the process already has at least one file open to a terminal.

SEE ALSO

ttyname(3C).

SUPPORT STATUS

Supported.

NAME

ctime, localtime, gmtime, asctime, tzset — convert date and time to string

SYNOPSIS

```
#include <time.h>

char *ctime (clock)
long *clock;

struct tm *localtime (clock)
long *clock;

struct tm *gmtime (clock)
long *clock;

char *asctime (tm)
struct tm *tm;

extern long timezone;

extern int daylight;

extern char *tzname[2];

void tzset ( )
```

DESCRIPTION

Ctime converts a long integer, pointed to by *clock*, representing the time in seconds since 00:00:00 GMT, January 1, 1970, and returns a pointer to a 26-character string in the following form (All the fields have constant width):

Sun Sep 16 01:03:52 1973\n\0

Localtime converts the long integer pointed to by *clock*, correcting for the time zone and possible Daylight Savings Time, and returns a *tm* structure.

Gmtime converts the long integer pointed to by *clock* directly to Greenwich Mean Time (GMT), which is the time the UNIX System uses, and returns a *tm* structure.

Asctime converts a *tm* structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

The external **long** variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, *timezone* is 5*60*60); the external variable *daylight* is non-zero if and only if the standard U.S.A. Daylight Savings Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended by the user.

If an environment variable named **TZ** is present, *asctime* uses the contents of the variable to override the default time zone. The value of **TZ** must be a three-letter time zone name, followed by a number representing the difference between local time and Greenwich Mean Time in hours, followed by an optional three-letter name for a daylight time zone. For example, the setting for New Jersey would be **EST5EDT**. Setting **TZ** changes the values of the external variables *timezone* and *daylight*.

In addition, the time zone names contained in the external variable

char *tzname[2] = { "EST", "EDT" };

are set from the environment variable **TZ**. The function *tzset* sets these external variables from **TZ**; *tzset* is called by *asctime* and may also be called explicitly by the user.

Note that **TZ** is set by default when the user logs on, to a value in the local /etc/profile file (see *profile*(4)).

TM STRUCTURE

Declarations of all the functions and externals, and the *tm* structure, are in the <*time.h*> header file. The structure declaration is:

```
struct tm {
        int tm_sec; /* seconds (0 - 59) */
        int tm_min; /* minutes (0 - 59) */
        int tm_hour; /* hours (0 - 23) */
        int tm_mday; /* day of month (1 - 31) */
        int tm_mon; /* month of year (0 - 11) */
        int tm_year; /* year − 1900 */
        int tm_wday; /* day of week (Sunday = 0) */
        int tm_yday; /* day of year (0 - 365) */
        int tm_isdst;
};
```

Tm_isdst is non-zero if Daylight Savings Time is in effect.

SEE ALSO

time(2), getenv(3C), profile(4), environ(5).

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – classify characters

SYNOPSIS

#include <ctype.h>

int isalpha (c)
int c;

. . .

DESCRIPTION

These macros classify character-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (−1). See *stdio*(3S).

isalpha	*c* is a letter.
isupper	*c* is an upper-case letter.
islower	*c* is a lower-case letter.
isdigit	*c* is a digit [0-9].
isxdigit	*c* is a hexadecimal digit [0-9], [A-F] or [a-f].
isalnum	*c* is an alphanumeric (letter or digit).
isspace	*c* is a space, tab, carriage return, new-line, vertical tab, or form-feed.
ispunct	*c* is a punctuation character (neither control nor alphanumeric).
isprint	*c* is a printing character, code 040 (space) through 0176 (tilde).
isgraph	*c* is a printing character, like *isprint* except false for space.
iscntrl	*c* is a delete character (0177) or an ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200.

DIAGNOSTICS

If the argument to any of these macros is not in the domain of the function, the result is undefined.

SEE ALSO

stdio(3S), ascii(5).

SUPPORT STATUS

Supported.

NAME

curses − screen functions with optimal cursor motion

SYNOPSIS

#include <cursesr2.h>

cc [options] files **−lcursesr2 −ltermcapr2** [libraries]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the *refresh* uses the routines to make the current screen look like the new one.

In order to initialize the routines, the routine *initscr* must be called before any of the other routines that deal with windows and screens are used.

The routine *endwin* should be called before exiting.

INPUT FILTERING

Routines are included in the library to map ASCII strings generated by terminal input keys or sequences of keystrokes to standard unused ASCII values so that applications would see a terminal independent input stream.

For example, function keys may be defined on a terminal without such keys by pressing the escape key followed by a digit. Input filtering would translate this sequence to a single ASCII code that could be recognized by the application.

By using the defined function key values in the termcap description of the terminal (see *termcap*(5)), several different terminal types could be supported with no application changes using input filtering.

Several default input translations are defined for use. The list below gives these definitions:

Function Keys 0 thru 9 are mapped to standard ASCII codes.

Backspace, Horizontal Tab, Cursor Movement keys (UP, DOWN, LEFT, RIGHT Arrow, HOME) are mapped to ASCII code values. See INPUT TRANSLATION SUMMARY below.

A set of functions calls in *curses* tests a character for a special key. The list below gives these functions:

isfunc(c) − the value of c is a function key
isbsp(c) − the value of c is a backspace key
istab(c) − the value of c is a tab key
isdarrow(c) − the value of c is a down arrow
isuarrow(c) − the value of c is a up arrow
islarrow(c) − the value of c is a left arrow
israrrow(c) − the value of c is a right arrow
ishome(c) − the value of c is a home Key

These function calls return a non-zero value if the condition tested for is true as do the standard C data typing functions (see *ctype*(3).

If a terminal capability description contains duplicate strings for two or more of the keys that are being filtered, only the ASCII code for the first is returned.

Values for many of the control sequence codes are reassigned. The assumption made here is that end user interfaces do not use these characters for input. Any interfaces that do use the chararacters work, but must be aware of the reassignment if the filtered keys are to be used.

INPUT TRANSLATION SUMMARY

The value in parentheses is a symbol defined to the ASCII value for the character which is available for comparison. TERMCAP Entry gives the terminal capability string id used in translating the key into the ASCII value listed.

Description	Ascii Character Returned	TERMCAP Entry
Function Code 0	0x10 (FK0)	k0
Function Code 1	0x11 (FK1)	k1
Function Code 2	0x12 (FK2)	k2
Function Code 3	0x13 (FK3)	k3
Function Code 4	0x14 (FK4)	k4
Function Code 5	0x15 (FK5)	k5
Function Code 6	0x16 (FK6)	k6
Function Code 7	0x17 (FK7)	k7
Function Code 8	0x18 (FK8)	k8
Function Code 9	0x19 (FK9)	k9
Backspace	0x08 (BSP)	kb
Hortizontal Tab	0x09 (HTB)	kt
Down Arrow	0x01 (DNA)	kd
UP Arrow	0x02 (UPA)	ku
Right Arrow	0x03 (RTA)	kr
Left Arrow	0x04 (LTA)	kl
Home	0x05 (HME)	kh

FUNCTIONS

addch(ch) — add a character to *stdscr*
addstr(str) — add a string to *stdscr*
box(win,vert,hor) — draw a box around a window
crmode() — set cbreak mode
clear() — clear *stdscr*
clearok(scr,boolf) — set clear flag for *scr*
clrtobot() — clear to bottom on *stdscr*
clrtoeol() — clear to end of line on *stdscr*
delch() — delete a character
deleteln() — delete a line
delwin(win) — delete *win*
echo() — set echo mode
endwin() — end window modes

erase() — erase *stdscr*
getch() — get a char through *stdscr*
getcap(name) — get terminal capability *name*
getstr(str) — get a string through *stdscr*
gettmode() — get tty modes
getyx(win,y,x) — get (y,x) coordinates; no status returned
inch() — get char at current (y,x) coordinates
initscr()-initialize screens
insch(c) — insert a char
insertln() — insert a line
leaveok(win,boolf) — set leave flag for *win*
longname(termbuf,name) — get long name from *termbuf*
move(y,x) — move to (y,x) on *stdscr*
mvcur(lasty,lastx,newy,newx) — actually move cursor
newwin(lns,cols,newy,newx) — create a new window
nl() — set newline mapping
nocrmode() — unset cbreak mode
noecho() — unset echo mode
nonl() — unset newline mapping
noraw() — unset raw mode
overlay(win1,win2) — overlay win1 on win2
overwrite(win1,win2) — overwrite win1 on top of win2
printw(fmt,arg1,arg2,...) — printf on *stdscr*
raw() — set raw mode
refresh() — make current screen look like *stdscr*
resetty() — reset tty flags to stored value
savetty() — stored current tty flags
scanw(fmt,arg1,arg2,...) — scanf through *stdscr*
scroll(win) — scroll *win* one line
scrollok(win,boolf) — set scroll flag
setterm(name) — set term variables for name
standend() — end standout mode
standout() — start standout mode
subwin(win,lns,cols,newy,newx) — create a subwindow
touchwin(win) — change all of *win*
unctrl(ch) — printable version of *ch*
waddch(win,ch) — add char to *win*
waddstr(win,str) — add string to *win*
wclear(win) — clear *win*
wclrtobot(win) — clear to bottom of *win*
wclrtoeol(win) — clear to end of line on *win*
wdelch(win,c) — delete char from *win*
wdeleteln(win) — delete line from *win*
werase(win) — erase *win*
wgetch(win) — get a char through *win*
wgetstr(win,str) — get a string through *win*
winch(win) — get char at current (y,x) in *win*
winsch(win,c) — insert char into *win*
winsertln(win) — insert line into *win*
wmove(win,y,x) — set current (y,x) coordinates on *win*
wprintw(win,fmt,arg1,arg2,...) — printf on *win*
wrefresh(win) — make screen look like *win*
wscanw(win,fmt,arg1,arg2,...) — scanf through *win*

wstandend(win) – end standout mode on *win*
wstandout(win) – start standout mode on *win*

RETURN STATUS

If a value of NULL is returned from the call to *initscr,* the memory cell *curses_err* contains the error status indicating the reason for the failure of *initscr*:

ST_OK
: No error condition, *initscr* worked.

ST_ERR01
: The TERM shell variable was not setup.

ST_ERR02
: Terminal type not found in TERMCAP file.

ST_ERR03
: No value for lines (*li*) in TERMCAP entry.

ST_ERR04
: No value for coloums (*co*) in TERMCAP entry.

SEE ALSO

termcap(5).

SUPPORT STATUS

Supported.

NAME

curses − CRT screen handling and optimization package

SYNOPSIS

#include <curses.h>
cc [options] files **−lcurses** [libraries]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. In order to initialize the routines, the routine *initscr* must be called before any of the other routines that deal with windows and screens are used. The routine *endwin* should be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented-programs want this) after calling *initscr* you should call *nonl; cbreak; noecho;*

The full curses interface permits manipulation of data structures called *windows* which can be thought of as two dimensional arrays of characters representing all or part of a CRT screen. A default window called **stdscr** is supplied, and others can be created with **newwin**. Windows are referred to by variables declared WINDOW *, the type WINDOW is defined in curses.h to be a C structure. These data structures are manipulated with functions described below, among which the most basic are **move**, and **addch**. (More general versions of these functions are included with names beginning with w, permitting you to specify a window. The routines not beginning with w affect **stdscr**.) Then *refresh* is called, telling the routines to make the users CRT screen look like **stdscr**.

Mini-Curses is a subset of curses which does not allow manipulation of more than one window. To invoke this subset, use −DMINICURSES as a cc option. This level is smaller and faster than full curses.

If the environment variable TERMINFO is defined, any program using curses checks for a local terminal definition before checking in the standard place. For example, if the standard place is **/usr/lib/terminfo**, and TERM is set to vt100, then normally the compiled file is found in **/usr/lib/terminfo/v/vt100**. (The v is copied from the first letter of vt100 to avoid creation of huge directories.) However, if TERMINFO is set to **/usr/mark/myterms**, curses first checks **/opusr/mark/myterms/v/vt100**, and if that fails, then checks **/usr/lib/terminfo/v/vt100**. This is useful for developing experimental definitions or when write permission in **/usr/lib/terminfo** is not available.

SEE ALSO

terminfo(4).

FUNCTIONS

Routines listed here may be called when using the full curses. Those marked with an asterisk may be called when using Mini-Curses.

addch(ch) − add a character to *stdscr* (like putchar) (wraps to next line at end of line)

addstr(str)* — calls addch with each character in *str*
attroff(attrs)* — turn off attributes named
attron(attrs)* — turn on attributes named
attrset(attrs)* — set current attributes to *attrs*
baudrate( )* — current terminal speed
beep( )* — sound beep on terminal
box(win, vert, hor) — draw a box around edges of *win vert* and *hor* are chars to use for vert. and hor. edges of box
clear( ) — clear *stdscr*
clearok(win, bf) — clear screen before next redraw of *win*
clrtobot( ) — clear to bottom of *stdscr*
clrtoeol( ) — clear to end of line on *stdscr*
cbreak( )* — set cbreak mode
delay_output(ms)* — insert ms millisecond pause in output
delch( ) — delete a character
deleteln( ) — delete a line
delwin(win) — delete *win*
doupdate( ) — update screen from all wnooutrefresh
echo( )* — set echo mode
endwin( )* — end window modes
erase( ) — erase *stdscr*
erasechar( ) — return user's erase character
fixterm( ) — restore tty to "in curses" state
flash( ) — flash screen or beep
flushinp( )* — throw away any typeahead
getch( )* — get a char from tty
getstr(str) — get a string through *stdscr*
gettmode( ) — establish current tty modes
getyx(win, y, x) — get (y, x) co-ordinates
has_ic( ) — true if terminal can do insert character
has_il( ) — true if terminal can do insert line
idlok(win, bf)* — use terminal's insert/delete line if bf != 0
inch( ) — get char at current (y, x) co-ordinates
initscr( )* — initialize screens
insch(c) — insert a char
insertln( ) — insert a line
intrflush(win, bf) — interrupts flush output if bf is TRUE
keypad(win, bf) — enable keypad input
killchar( ) — return current user's kill character
leaveok(win, flag) — OK to leave cursor anywhere after refresh if flag!=0 for *win*, otherwise cursor must be left at current position.
longname( ) — return verbose name of terminal
meta(win, flag)* — allow meta characters on input if flag != 0
move(y, x)* — move to (y, x) on *stdscr*
mvaddch(y, x, ch) — move(y, x) then addch(ch)
mvaddstr(y, x, str) — similar...
mvcur(oldrow, oldcol, newrow, newcol) — low level cursor motion
mvdelch(y, x)   like delch, but move(y, x) first
mvgetch(y, x) — etc.
mvgetstr(y, x)
mvinch(y, x)
mvinsch(y, x, c)

mvprintw(y, x, fmt, args)
mvscanw(y, x, fmt, args)
mvwaddch(win, y, x, ch)
mvwaddstr(win, y, x, str)
mvwdelch(win, y, x)
mvwgetch(win, y, x)
mvwgetstr(win, y, x)
mvwin(win, by, bx)
mvwinch(win, y, x)
mvwinsch(win, y, x, c)
mvwprintw(win, y, x, fmt, args)
mvwscanw(win, y, x, fmt, args)
newpad(nlines, ncols) — create a new pad with given dimensions
newterm(type, ofd, ifd) — set up new terminal of given type to output on ofd and input on ifd
newwin(lines, cols, begin_y, begin_x) — create a new window
nl( )* — set newline mapping
nocbreak( )* — unset cbreak mode
nodelay(win, bf) — enable nodelay input mode through getch
noecho( )* — unset echo mode
nonl( )* — unset newline mapping
noraw( )* — unset raw mode
overlay(win1, win2) — overlay win1 on win2
overwrite(win1, win2) — overwrite win1 on top of win2
pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol) — like prefresh but with no output until doupdate called
prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol) — refresh from pad starting with given upper left corner of pad with output to given portion of screen
printw(fmt, arg1, arg2, ...) — printf on *stdscr*
raw( )* — set raw mode
refresh( )* — make current screen look like *stdscr*
resetterm( )* — set tty modes to "out of curses" state
resetty( )* — reset tty flags to stored value
saveterm( )* — save current modes as "in curses" state
savetty( )* — store current tty flags
scanw(fmt, arg1, arg2, ...) — scanf through *stdscr*
scroll(win) — scroll *win* one line
scrollok(win, flag) — allow terminal to scroll if flag != 0
set_term(new) — now talk to terminal new
setscrreg(t, b) — set user scrolling region to lines t through b
setterm(type) — establish terminal with given type
setupterm(term, filenum, errret)
standend( )* — clear standout mode attribute
standout( )* — set standout mode attribute
subwin(win, lines, cols, begin_y, begin_x) — create a subwindow
touchwin(win) — change all of *win*
traceoff( ) — turn off debugging trace output
traceon( ) — turn on debugging trace output
typeahead(fd) — use file descriptor fd to check typeahead
unctrl(ch)* — printable version of *ch*
waddch(win, ch) — add char to *win*

waddstr(win, str) — add string to *win*
wattroff(win, attrs) — turn off *attrs* in *win*
wattron(win, attrs) — turn on *attrs* in *win*
wattrset(win, attrs) — set attrs in *win* to *attrs*
wclear(win) — clear *win*
wclrtobot(win) — clear to bottom of *win*
wclrtoeol(win) — clear to end of line on *win*
wdelch(win, c) — delete char from *win*
wdeleteln(win) — delete line from *win*
werase(win) — erase *win*
wgetch(win) — get a char through *win*
wgetstr(win, str) — get a string through *win*
winch(win) — get char at current (y, x) in *win*
winsch(win, c) — insert char into *win*
winsertln(win) — insert line into *win*
wmove(win, y, x) — set current (y, x) co-ordinates on *win*
wnoutrefresh(win) — refresh but no screen output
wprintw(win, fmt, arg1, arg2, ...) — printf on *win*
wrefresh(win) — make screen look like *win*
wscanw(win, fmt, arg1, arg2, ...) — scanf through *win*
wsetscrreg(win, t, b) — set scrolling region of *win*
wstandend(win) — clear standout attribute in *win*
wstandout(win) — set standout attribute in *win*

TERMINFO LEVEL ROUTINES

These routines should be called by programs wishing to deal directly with the terminfo database. Due to the low level of this interface, it is discouraged. Initially, *setupterm* should be called. This defines the set of terminal dependent variables defined in terminfo(4). The include files <curses.h> and <term.h> should be included to get the definitions for these strings, numbers, and flags. Parmeterized strings should be passed through *tparm* to instantiate them. All terminfo strings (including the output of tparm) should be printed with *tputs* or *putp* . Before exiting, *resetterm* should be called to restore the tty modes. (Programs desiring shell escapes or suspending with control Z can call *resetterm* before the shell is called and *fixterm* after returning from the shell.)

fixterm( ) — restore tty modes for terminfo use (called by setupterm)
resetterm( ) — reset tty modes to state before program entry
setupterm(term, fd, rc) — read in database(see NOTE).
tparm(str, p1, p2, ..., p9) — instantiate string str with parms p_i.
tputs(str, affcnt, putc) — apply padding info to string *str*. *affcnt* is the number of lines affected, or 1 if not applicable. *Putc* is a putchar-like function to which the characters are passed, one at a time.
putp(str) — handy function that calls tputs (str, 1, putchar)
vidputs(attrs, putc) — output the string to put terminal in video attribute mode *attrs*, which is any combination of the attributes listed below. Chars are passed to putchar-like function *putc*.
vidattr(attrs) — Like vidputs but outputs through putchar

NOTE: Terminal type is the character string *term*, all output is to UNIX system file descriptor *fd*. A status value is returned in the integer pointed to by *rc*: 1 is normal. The simplest call would be *setupterm(0, 1, 0)* which uses all defaults.

TERMCAP COMPATIBILITY ROUTINES

These routines were included as a conversion aid for programs that use termcap. Their parameters are the same as for termcap. They are emulated using the *terminfo* database. They may go away at a later date.

tgetent(bp, name) — look up termcap entry for name
tgetflag(id) — get boolean entry for id
tgetnum(id) — get numeric entry for id
tgetstr(id, area) — get string entry for id
tgoto(cap, col, row) — apply parms to given cap
tputs(cap, affcnt, fn) — apply padding to cap calling fn as putchar

ATTRIBUTES

The following video attributes can be passed to the functions *attron,attroff,attrset*.

A_STANDOUT	Terminal's best highlighting mode
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_BLANK	Blanking (invisible)
A_PROTECT	Protected
A_ALTCHARSET	Alternate character set

FUNCTION KEYS

The following function keys might be returned by *getch* if *keypad* has been enabled. Note that not all of these are currently supported, due to lack of definitions in *terminfo* or the terminal not transmitting a unique code when the key is pressed.

Name	*Value*	*Key name*
KEY_BREAK	0401	break key (unreliable)
KEY_DOWN	0402	The four arrow keys ...
KEY_UP	0403	
KEY_LEFT	0404	
KEY_RIGHT	0405	...
KEY_HOME	0406	Home key (upward+left arrow)
KEY_BACKSPACE	0407	backspace (unreliable)
KEY_F0	0410	Function keys. Space for 64 is reserved.
KEY_F(n)	(KEY_F0+(n))	Formula for fn.
KEY_DL	0510	Delete line
KEY_IL	0511	Insert line
KEY_DC	0512	Delete character
KEY_IC	0513	Insert char or enter insert mode
KEY_EIC	0514	Exit insert char mode

KEY_CLEAR	0515	Clear screen
KEY_EOS	0516	Clear to end of screen
KEY_EOL	0517	Clear to end of line
KEY_SF	0520	Scroll 1 line forward
KEY_SR	0521	Scroll 1 line backwards (reverse)
KEY_NPAGE	0522	Next page
KEY_PPAGE	0523	Previous page
KEY_STAB	0524	Set tab
KEY_CTAB	0525	Clear tab
KEY_CATAB	0526	Clear all tabs
KEY_ENTER	0527	Enter or send (unreliable)
KEY_SRESET	0530	soft (partial) reset (unreliable)
KEY_RESET	0531	reset or hard reset (unreliable)
KEY_PRINT	0532	print or copy
KEY_LL	0533	home down or bottom (lower left)

WARNING

The plotting library *plot*(3X) and the curses library *curses*(3X) both use the names erase() and move(). The curses versions are macros. If you need both libraries, put the *plot*(3X) code in a different source file than the *curses*(3X) code, and/or #undef move() and erase() in the *plot*(3X) code.

SUPPORT STATUS

Supported.

NAME

cuserid − get character login name of the user

SYNOPSIS

#include <stdio.h>

char *cuserid (s)
char *s;

DESCRIPTION

Cuserid generates a character-string representation of the login name of the owner of the current process. If *s* is a NULL pointer, *cuserid* generates this representation in an internal static area, and returns its address. Otherwise, *cuserid* assumes that *s* points to an array of at least **L_cuserid** characters and leaves the representation in this array. The constant **L_cuserid** is defined in the **<stdio.h>** header file.

DIAGNOSTICS

If the login name cannot be found, *cuserid* returns a NULL pointer, and, if *s* is not a NULL pointer, places a null character (\0) at *s[0]*.

SEE ALSO

getlogin(3C), getpwent(3C).

SUPPORT STATUS

Supported.

NAME

dial – establish an out-going terminal line connection

SYNOPSIS

#include <dial.h>

int dial (call)
CALL *call;

void undial (fd)
int fd;

DESCRIPTION

Dial returns a file-descriptor for a terminal line open for read/write. The argument to *dial* is a CALL structure defined in the *<dial.h>* header file.

When finished with the terminal line, the calling program must invoke *undial* to release the semaphore that has been set during the allocation of the terminal device.

The CALL typedef in the *<dial.h>* header file is:

```
typedef struct {
        struct termio *attr;  /* pointer to termio attribute struct */
        int                   baud;
        int                   speed;
        char                  *line;
        char                  *telno;
        int                   modem;
} CALL;
```

The CALL element *speed* is intended only for use with an outgoing dialed call, in which case its value should be either 300 or 1200 to identify the 113A modem, or the high or low speed setting on the 212A modem.

The CALL element *baud* is for the desired transmission baud rate.

For example, one might set *baud* to 110 and *speed* to 300 (or 1200).

If the desired terminal line is a direct line, a string pointer to its device-name should be assigned to the *line* element in the CALL structure. Valid values for terminal device names are kept in the *L-devices* file. In this case, the value of the *baud* element need not be specified as it is determined from the *L-devices* file.

Telno is a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of symbols described on the *acu*(7). The termination symbol is supplied by the *dial* function, and should not be included in the *telno* string passed to *dial* in the CALL structure.

The CALL element *modem* specifies modem control for direct lines. This element should be non-zero if modem control is required.

The CALL element *attr* is a pointer to a *termio* structure, as defined in the *termio.h* header file. A NULL value for this pointer element may be passed to the *dial* function, but if such a structure is included, the elements specified in it are set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

FILES

/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..*tty-device*

SEE ALSO

uucp(1C), alarm(2), read(2), write(2), termio(7).

DIAGNOSTICS

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the *<dial.h>* header file.

```
INTRPT    -1  /* interrupt occured */
D_HUNG    -2  /* dialer hung (no return from write) */
NO_ANS    -3  /* no answer within 10 seconds */
ILL_BD    -4  /* illegal baud-rate */
A_PROB    -5  /* acu problem (open() failure) */
L_PROB    -6  /* line problem (open() failure) */
NO_Ldv    -7  /* can't open LDEVS file */
DV_NT_A   -8  /* requested device not available */
DV_NT_K   -9  /* requested device not known */
NO_BD_A   -10 /* no device available at requested baud */
NO_BD_K   -11 /* no device known at requested baud */
```

WARNINGS

Including the <dial.h> header file automatically includes the **<termio.h> header file.**

Dial(3C) uses <stdio.h>, and thus increases the size of programs not otherwise using standard I/O, more than might be expected.

RESTRICTIONS

An *alarm*(2) system call for 3600 seconds is made (and caught) within the *dial* module for the purpose of *touching* the *LCK..* file. This call constitutes the device allocation semaphore for the terminal device. Otherwise, *uucp*(1C) may simply delete the *LCK..* entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a *read*(2) or *write*(2) system call, causing an apparent error return. If the user program expects to execute for an hour or more, error returns from *reads* should be checked for **(errno==EINTR)**, and the *read* possibly reissued.

SUPPORT STATUS

Not supported.

NAME

dim, ddim, idim − positive difference intrinsic functions

SYNOPSIS

integer a1,a2,a3
a3 = idim(a1,a2)

real a1,a2,a3
a3 = dim(a1,a2)

double precision a1,a2.,a3
a3 = ddim(a1,a2)

DESCRIPTION

These functions return a1−a2 if a1 is greater than a2 or 0 if a1 is less than or equal to a2.

SUPPORT STATUS

Not supported.

NAME

dprod — double precision product intrinsic function

SYNOPSIS

real a1,a2
double precision a3
a3 = dprod (a1,a2)

DESCRIPTION

Dprod returns the double precision product of its real arguments.

SUPPORT STATUS

Not supported.

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS

double drand48 ( )

double erand48 (xsubi)
unsigned short xsubi[3];

long lrand48 ( )

long nrand48 (xsubi)
unsigned short xsubi[3];

long mrand48 ( )

long jrand48 (xsubi)
unsigned short xsubi[3];

void srand48 (seedval)
long seedval;

unsigned short *seed48 (seed16v)
unsigned short seed16v[3];

void lcong48 (param)
unsigned short param[7];

DESCRIPTION

These functions generate pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions *drand48* and *erand48* return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions *lrand48* and *nrand48* return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

Functions *mrand48* and *jrand48* return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

Functions *srand48, seed48* and *lcong48* are initialization entry points, one of which should be invoked before either *drand48, lrand48* or *mrand48* is called. (Although it is not recommended practice, constant default initializer values are supplied automatically if *drand48, lrand48* or *mrand48* is called without a prior call to an initialization entry point.) Functions *erand48, nrand48* and *jrand48* do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\mathrm{mod}\ m} \qquad n \geq 0.$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48* has been invoked, the multiplier value a and the addend value c are given by

$a = 5DEECE66D_{16} = 273673163155_8$
$c = B_{16} = 13_8.$

The value returned by any of the functions *drand48, erand48, lrand48, nrand48, mrand48* or *jrand48* is computed by first generating the next 48-bit X_i in the sequence. Then the function copies the appropriate number of bits, according to the type of data item to be returned, from the high-order (leftmost) bits of X_i and transforms them into the returned value.

The functions *drand48, lrand48* and *mrand48* store the last 48-bit X_i generated in an internal buffer; that is why they must be initialized prior to being invoked.

The functions *erand48, nrand48* and *jrand48* require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument.

By using different arguments, functions *erand48, nrand48* and *jrand48* allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, i.e., the sequence of numbers in each stream does *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function *srand48* sets the high-order 32 bits of X_i to the 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function *seed48* sets the value of X_i to the 48-bit value specified in the argument array. In addition, *seed48* copies the previous value of X_i into a 48-bit internal buffer, used only by *seed48*, and returns a pointer to this buffer. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time: use the pointer to access and store the last X_i value, and then use this value to reinitialize via *seed48* when the program is restarted.

The initialization function *lcong48* allows the user to specify the initial X_i, the multiplier value a, and the addend value c. Argument array elements *param[0-2]* specify X_i, *param[3-5]* specify the multiplier a, and *param[6]* specifies the 16-bit addend c. After *lcong48* has been called, a subsequent call to either *srand48* or *seed48* restores the *standard* multiplier and addend values, a and c, specified on the previous page.

SEE ALSO
rand(3C).

SUPPORT STATUS
Supported.

NAME

ecvt, fcvt, gcvt — convert floating-point number to string

SYNOPSIS

char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt (value, ndigit, buf)
double value;
char *buf;

DESCRIPTION

Ecvt converts *value* to a null-terminated string of *ndigit* digits and returns a pointer to that string. The low-order digit is rounded.

The decimal point is not included in the returned string. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt*; a negative number indicates the decimal point belongs to the left of the returned digits.

If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

Fcvt is identical to *ecvt*, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *ndigit*.

Gcvt converts the *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. *Gcvt* attempts to produce *ndigit* significant digits in Fortran F-format if possible, otherwise E-format, ready for printing. A minus sign, if there is one, or a decimal point is included as part of the returned string. Trailing zeros are suppressed.

SEE ALSO

printf(3S).

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

end, etext, edata — last locations in program

SYNOPSIS

extern end;
extern etext;
extern edata;

DESCRIPTION

The address of *etext* is the first address above the program text.

The address of *edata* is the first address above the initialized data region.

The address of *end* is the first address above the unitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with *end*, but the program break may be reset by the routines of *brk*(2), *malloc*(3C), standard input/output (*stdio*(3S)), the profile (**−p**) option of *cc*(1), and so on. Thus, the current value of the program break should be determined by **sbrk(0)** (see *brk*(2)).

SEE ALSO

brk(2), malloc(3C), stdio(3S), cc(1).

SUPPORT STATUS

Supported.

NAME

erf, erfc — error function and complementary error function

SYNOPSIS

```
#include <math.h>

double erf (x)
double x;

double erfc (x)
double x;
```

DESCRIPTION

Erf returns the error function of x, defined as

$$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt.$$

Erfc returns 1.0 − *erf(x)*. If *erf(x)* is called for large x and the result subtracted from 1.0, loss of accuracy is extreme: for example, for $x = 5$, 12 places are lost. *Erfc* retains much of the otherwise lost accuracy.

SEE ALSO

exp(3M).

SUPPORT STATUS

Supported.

NAME

exp, dexp, cexp, log, alog, dlog, clog, log10, alog10, dlog10, sqrt, dsqrt, csqrt — Fortran exponential, logarithm, square root intrinsic functions

SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2 = exp(r1)

dp2 = dexp(dp1)
dp2 = exp(dp1)

cx2 = cexp(cx1)
cx2 = exp(cx1)

r2 = alog(r1)
r2 = log(r1)

dp2 = dlog(dp1)
dp2 = log(dp1)

cx2 = clog(cx1)
cx2 = log(cx1)

r2 = alog10(r1)
r2 = log10(r1)

dp2 = dlog10(dp1)
dp2 = log10(dp1)

r2 = sqrt(r1)

dp2 = dsqrt(dp1)
dp2 = sqrt(dp1)

cx2 = csqrt(cx1)
cx2 = sqrt(cx1)
```

DESCRIPTION

Exp, the generic exponential function, returns the exponential function, e^x, of its argument, the returned value is the same type as its argument.

Dexp returns the double-precision exponential function of its double-precision argument.

Cexp returns the complex exponential function of its complex argument.

Log, the generic natural logarithm function, returns the natural logarithm of its argument; the returned value is the same type as the argument.

Alog returns the real natural logarithm of its real argument.

Dlog returns the double-precision natural logarithm of its double-precision argument.

Clog returns the complex logarithm of its complex argument.

Log10, the generic common logarithm function, returns the common logarithm of its argument; the returned value is the same type as the argument.

Alog10 returns the real common logarithm of its real argument.

Dlog10 returns the double-precision common logarithm of its double-precision argument. *Sqrt*, the generic square root function, returns the square root of its argument; the type of the returned value is the same as the argument.

Dsqrt returns the double-precision square root of its double-precision arguement.

Csqrt returns the complex square root of its complex argument.

SEE ALSO

exp(3M).

SUPPORT STATUS

Not supported.

NAME

exp, log, log10, pow, sqrt — exponential, logarithm, power, square root functions

SYNOPSIS

#include <math.h>

```
double exp (x)
double x;

double log (x)
double x;

double log10 (x)
double x;

double pow (x, y)
double x, y;

double sqrt (x)
double x;
```

DESCRIPTION

Exp returns e^x.

Log returns the natural logarithm of x. The value of x must be positive.

Log10 returns the logarithm base ten of x. The value of x must be positive.

Pow returns x^y. The values of x and y may not both be zero. If x is non-positive, y must be an integer.

Sqrt returns the square root of x. The value of x may not be negative.

DIAGNOSTICS

Exp returns HUGE when the correct value would overflow, and sets *errno* to ERANGE.

Log and *log10* return 0 and set *errno* to EDOM when x is non-positive. An error message is printed on the standard error output.

Pow returns 0 and sets *errno* to EDOM when x is non-positive and y is not an integer, or when x and y are both zero. In these cases *pow* prints a message indicating DOMAIN error on the standard error output.

When the correct value for *pow* would overflow, *pow* returns HUGE and sets *errno* to ERANGE.

When x is negative, *sqrt* returns 0, sets *errno* to EDOM, and prints a message indicating DOMAIN error on the standard error output.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

hypot(3M), matherr(3M).

SUPPORT STATUS

Supported.

NAME

fclose, fflush — close or flush a stream

SYNOPSIS

#include <stdio.h>

int fclose (stream)
FILE *stream;

int fflush (stream)
FILE *stream;

DESCRIPTION

Fclose writes out any buffered data for the named *stream* and closes that *stream*. *Fclose* is performed automatically for all open files upon calling *exit*(2).

Fflush writes any buffered data for the named *stream* to that file. The *stream* remains open.

DIAGNOSTICS

These functions return 0 for success, and EOF if any error (such as trying to write to a file that has not been opened for writing) is detected.

SEE ALSO

close(2), exit(2), fopen(3S), setbuf(3S).

SUPPORT STATUS

Supported.

NAME

ferror, feof, clearerr, fileno — stream status inquiries

SYNOPSIS

#include <stdio.h>

int feof (stream)
FILE
***stream;**

int ferror (stream)
FILE
***stream;**

void clearerr (stream)
FILE
***stream;**

int fileno(stream)
FILE
***stream;**

DESCRIPTION

Feof returns non-zero when EOF has previously been detected reading the named input *stream*, otherwise *feof* returns zero.

Ferror returns non-zero when an I/O error has previously occurred reading from or writing to the named *stream*, otherwise *ferror* returns zero.

Clearerr resets the error indicator and EOF indicator to zero on the named *stream*.

Fileno returns the integer file descriptor associated with the named *stream*; see *open*(2).

NOTE

All these functions are implemented as macros; they cannot be declared or redeclared.

SEE ALSO

open(2), fopen(3S).

SUPPORT STATUS

Supported.

NAME

fltoieee, dbtoieee, fltomit, dbtomit — float format conversions

SYNOPSIS

```
double dbtoieee (dbnum)
double dbnum;

double dbtomit (dbnum)
double dbnum;

#include <fp.h>

fp fltoieee (flnum)
fp flnum;

fp fltomit (flnum)
fp flnum;
```

DESCRIPTION

These routines are used for converting float and double values to the Release 3 (IEEE) and the Release 2 (MIT) floating point formats.

Dbtoieee converts a double value in Release 2 to Release 3 format; *dbtomit* converts Release 3 to Release 2.

Fltoieee and *fltomit* do the same for float values. Note that the float value must first be moved to a variable of type *fp* to inhibit float to double conversion.

In *dbtomit* and *fltomit* , values are checked before conversion to see if they are the special IEEE constants for infinity and Not-A-Number. A value of infinity is returned and *errno* is set to *ERANGE*.

SUPPORT STATUS

Supported.

NAME

floor, ceil, fmod, fabs — floor, ceiling, remainder, absolute value functions

SYNOPSIS

```
#include <math.h>

double floor (x)
double x;

double ceil (x)
double x;

double fmod (x, y)
double x, y;

double fabs (x)
double x;
```

DESCRIPTION

Floor returns the largest integer (as a double-precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the floating-point remainder of the division of x by y. It returns zero if y is zero or if x/y would overflow; otherwise the number f with the same sign as x, such that $x = iy + f$ for some integer i, and $|f| < |y|$.

Fabs returns the absolute value of x, $|x|$.

SEE ALSO

abs(3C).

SUPPORT STATUS

Supported.

NAME

fopen, freopen, fdopen − open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen (file-name, type)
char *file-name, *type;

FILE *freopen (file-name, type, stream)
char *file-name, *type;
FILE *stream;

FILE *fdopen (fildes, type)
int fildes;
char *type;

DESCRIPTION

Fopen opens the file named by *file-name* and associates a *stream* with it. *Fopen* returns a pointer to the FILE structure associated with the *stream*.

File-name points to a character string that contains the name of the file to be opened.

Type is a character string having one of the following values:

r	open for reading
w	truncate or create for writing
a	append; open for writing at end of file, or create for writing
r+	open for update (reading and writing)
w+	truncate or create for update
a+	append; open or create for update at end-of-file

Freopen substitutes the named file in place of the open *stream*. The original *stream* is closed, regardless of whether the open ultimately succeeds. *Freopen* returns a pointer to the FILE structure associated with *stream*.

Freopen is typically used to attach the preopened *streams* associated with **stdin**, **stdout** and **stderr** to other files.

Fdopen associates a *stream* with a file descriptor obtained from *open*, *dup*, *creat*, or *pipe*(2), which opens files but does not return pointers to a FILE structure *stream* which are necessary input for many of the section 3S library routines. The *type* of *stream* must agree with the mode of the open file.

When a file is opened for update, both reading and writing may be performed on the resulting *stream*. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end-of-file.

When a file is opened for append (i.e., when *type* is *a* or *a+*) it is impossible to overwrite information already in the file. *Fseek* may be used to reposition the file pointer to any position in the file, but when output is written to the file the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process writes to the file without destroying output being written by the other; the output from the two processes is intermixed in the file in the order in which it is written.

SEE ALSO

creat(2), dup(2), open(2), pipe(2), fclose(3S), fseek(3S).

DIAGNOSTICS

Fopen and *freopen* return a NULL pointer on failure.

SUPPORT STATUS

Supported.

NAME

fread, fwrite — binary input/output

SYNOPSIS

#include <stdio.h>

```
int fread (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;

int fwrite (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

DESCRIPTION

Fread copies *nitems* items of data from the named input *stream* into an array beginning at *ptr*. An item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. *Fread* stops appending bytes when an end-of-file or error condition is encountered while reading *stream*, or when *nitems* items have been read. *Fread* leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. *Fread* does not change the contents of *stream*.

Fwrite appends at most *nitems* items of data from the the array pointed to by *ptr* to the named output *stream*. *Fwrite* stops appending when it has appended *nitems* items of data or when an error condition is encountered on *stream*. *Fwrite* does not change the contents of the array pointed to by *ptr*.

The variable *size* is typically *sizeof(*ptr)* where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char* it should be cast into a pointer to *char*.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S).

RETURN VALUE

Fread and *fwrite* return the number of items read or written. If *nitems* is non-positive, no characters are read or written and 0 is returned by both *fread* and *fwrite*.

SUPPORT STATUS

Supported.

NAME

frexp, ldexp, modf — manipulate parts of floating-point numbers

SYNOPSIS

double frexp (value, eptr)
double value;
int *eptr;

double ldexp (value, exp)
double value;
int exp;

double modf (value, iptr)
double value, *iptr;

DESCRIPTION

Every non-zero number can be written uniquely as $x * 2^n$, where the *mantissa* (fraction) x is in the range $0.5 \leq |x| < 1.0$, and the *exponent* n is an integer. *Frexp* returns the mantissa of a double precision *value*, and stores the exponent indirectly in the location pointed to by *eptr*.

Ldexp returns the quantity *value* $* \; 2^{exp}$.

Modf returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

RETURN VALUE

If *ldexp* would cause overflow, *ldexp* returns HUGE and sets *errno* to ERANGE.

If *ldexp* would cause underflow, *ldexp* returns zero and sets *errno* to ERANGE.

SUPPORT STATUS

Supported.

NAME

fseek, rewind, ftell — reposition a file pointer in a stream

SYNOPSIS

```
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position depends on the values of *ptrname* and the signed distance *offset*:

If *ptrname* is 0, the new position is *offset* bytes from the beginning. (*Offset* should be positive.)

If *ptrname* is 1, the new position is *offset* bytes from the current position.

If *ptrname* is 2, the new position is *offset* bytes from the end of the file. (*Offset* should be negative.)

Rewind(*stream*) is equivalent to *fseek*(*stream*, 0L, 0), except that no value is returned.

Fseek and *rewind* undo any effects of *ungetc*(3S).

After *fseek* or *rewind*, the next operation on a file opened for update may be either input or output.

Ftell returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

SEE ALSO

lseek(2), fopen(3S), popen(3S), ungetc(3S).

RETURN VALUE

The following are the return values for *fseek*:

zero	successful completion
non-zero	improper seek request

An improper seek is for example, an *fseek* to an unopened file (no preceding *fopen*); in particular, *fseek* may not be used on a terminal, or on a file opened via *popen*(3S).

WARNING

On the UNIX system the *offset* may be an arithmetic expression using the value returned by *ftell*. For portability to non-UNIX systems, however, an absolute offset should be used as a parameter to *fseek*, because the value returned by *ftell* may not be measured in bytes.

SUPPORT STATUS

Supported.

NAME

ftw — walk a file tree

SYNOPSIS

#include <ftw.h>

int ftw (path, fn, depth)
char *path;
int (*fn) ( );
int depth;

DESCRIPTION

Ftw recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, *ftw* calls the function *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a stat structure (see *stat*(2)) containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> header file, are:

FTW_F a file
FTW_D a directory
FTW_DNR a directory that cannot be read
FTW_NS an object for which *stat* could not be executed successfully.

If the integer is FTW_DNR, descendants of that directory are not processed. If the integer is FTW_NS, the stat structure contains garbage. For example, a file in a directory with read but without execute (search) permission causes FTW_NS to be passed to *fn*.

Ftw visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within *ftw* (such as an I/O error).

Ftw uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *Depth* must not be greater than the number of file descriptors currently available for use. *Ftw* runs more quickly if *depth* is at least as large as the number of levels in the tree.

RETURN VALUE

0 the tree is exhausted
nonzero traversal stopped due to nonzero return by *fn*
−1 error; *errno* indicatates error type

SEE ALSO

stat(2), malloc(3C).

RESTRICTIONS

Because *ftw* is recursive, it may terminate with a memory fault when applied to very deep file structures.

Ftw uses *malloc*(3C) to allocate dynamic storage during its operation. If *ftw* is forcibly terminated, such as by *longjmp* executed by *fn* or an interrupt routine, *ftw* has no chance to free that storage, so the storage remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

SUPPORT STATUS

Supported.

NAME

int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char — explicit Fortran type conversion

SYNOPSIS

integer i, j
real r, s
double precision dp, dq
complex cx
double complex dcx
character*1 ch

i = int(r)
i = int(dp)
i = int(cx)
i = int(dcx)
i = ifix(r)
i = idint(dp)

r = real(i)
r = real(dp)
r = real(cx)
r = real(dcx)
r = float(i)
r = sngl(dp)

dp = dble(i)
dp = dble(r)
dp = dble(cx)
dp = dble(dcx)

cx = cmplx(i)
cx = cmplx(i, j)
cx = cmplx(r)
cx = cmplx(r, s)
cx = cmplx(dp)
cx = cmplx(dp, dq)
cx = cmplx(dcx)

dcx = dcmplx(i)
dcx = dcmplx(i, j)
dcx = dcmplx(r)
dcx = dcmplx(r, s)
dcx = dcmplx(dp)
dcx = dcmplx(dp, dq)
dcx = dcmplx(cx)

i = ichar(ch)
ch = char(i)

DESCRIPTION

These functions convert one data type to another.

int

Int converts to *integer* form its *real, double precision, complex*, or *double complex* argument. If the argument is *real* or *double precision*, int returns the integer whose magnitude is the largest integer that does not exceed the magnitude of the argument and whose sign is the same as the sign of the argument (truncation). If the argument is complex, *int* returns the truncated real portion of the complex number.

Ifix and **idint** convert only *real* and *double precision* arguments respectively.

real, float, sngl

Real converts to *real* form an *integer, double precision, complex*, or *double complex* argument. If the argument is *double precision* or *double complex*, *real* keeps as much precision as possible. If the argument is complex, *real* returns the real part of the complex number.

Float and **sngl** convert only *integer* and *double precision* arguments respectively.

dble

Dble converts any *integer, real, complex*, or *double complex* argument to *double precision* form. If the argument is of a complex type, the real part is returned.

cmplx, dcmplx

Cmplx converts its *integer, real, double precision*, or *double complex* argument(s) to *complex* form.

Dcmplx converts to *double complex* form its *integer, real, double precision*, or *complex* argument(s).

Either one or two arguments may be supplied to **cmplx** and **dcmplx** . If there is only one argument, it is taken as the real part of the complex type and a imaginary part of zero is supplied. If two arguments are supplied, the first is taken as the real part and the second as the imaginary part.

ichar, char

Ichar converts from a character to an integer depending on the position of the character in the collating sequence.

Char returns the character in the *i*th position in the processor collating sequence where *i* is the supplied argument.

For a processor capable of representing *n* characters,

ichar(char(i)) = i for 0 <= i < *n*, and

char(ichar(ch)) = ch for any representable character *ch*.

SUPPORT STATUS

Not supported.

NAME

gamma − log gamma function

SYNOPSIS

#include <math.h>

extern int signgam;

double gamma (x)
double x;

DESCRIPTION

Gamma returns $\ln(|\Gamma(x)|)$, where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t} t^{x-1} dt$.

Gamma returns the sign of $\Gamma(x)$ in the external integer *signgam*. The argument x may not be a non-positive integer.

EXAMPLE

The following C program fragment might be used to calculate Γ:

```
if ((y = gamma(x)) > LOGHUGE)
        error( );
y = signgam * exp(y);
```

where LOGHUGE is the least value that causes *exp*(3M) to return a range error.

RETURN VALUE

For non-negative integer arguments, *gamma* returns HUGE, and sets *errno* to EDOM. *Gamma* prints a message indicating DOMAIN error on the standard error output.

If the correct value would overflow, *gamma* returns HUGE and sets *errno* to ERANGE.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

exp(3M), matherr(3M), values(5).

SUPPORT STATUS

Supported.

NAME

getarg − return Fortran command-line argument

SYNOPSIS

character*N c
integer i

call getarg(i, c)

DESCRIPTION

Getarg returns the *i*-th command-line argument of the current process. Thus, if a program were invoked using

function arg1 arg2 arg3

getarg(2, c) would return the string *arg2* in the character variable *c*.

SEE ALSO

getopt(3C).

SUPPORT STATUS

Not supported.

NAME

getc, getchar, fgetc, getw — get character or word from stream

SYNOPSIS

```
#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar ()

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;
```

DESCRIPTION

Getc

Getc returns the next character (i.e. byte) from the named input *stream*. It also moves the file pointer, if defined, ahead one character in *stream*. *Getc* is a macro and so cannot be used if a function is necessary; for example one cannot have a function pointer point to it.

Getchar

Getchar returns the next character from the standard input stream, *stdin*. Like *getc*, *getchar* is a macro.

Fgetc

Fgetc performs the same function as *getc*, but is a genuine function. *Fgetc* runs more slowly than *getc*, but takes less space per invocation.

Getw

Getw returns the next word (i.e. integer) from the named input *stream*. The size of a word varies from machine to machine.

Getw returns the constant **EOF** upon end-of-file or error, but as that is a valid integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*.

Getw increments the associated file pointer, if defined, to point to the next word.

Getw assumes no special alignment in the file.

SEE ALSO

fclose(3S), ferror(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S).

RETURN VALUE

These functions return the integer constant **EOF** at end-of-file or upon an error.

RESTRICTIONS

Because it is implemented as a macro, *getc* does not treat a *stream* argument with side effects correctly. In particular, getc(*f++) does not work correctly because the string *f++ is substituted, not the value. The function *fgetc* should be used instead when side effects are used.
Because of possible differences in word length and byte ordering,

files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor.

SUPPORT STATUS

Supported.

NAME

getcwd — get path-name of current working directory

SYNOPSIS

char *getcwd (buf, size)
char *buf;
int size;

DESCRIPTION

Getcwd returns a pointer to the current directory path-name. *Buf* is the location in which the pathname is to be stored. *Size* must be at least two greater than the length of the path-name to be returned.

If *buf* is a NULL pointer, *getcwd* obtains *size* bytes of space using *malloc*(3C). In this case, the pointer returned by *getcwd* may be used as the argument in a subsequent call to *free.*

Getcwd is implemented by using *popen*(3S) to pipe the output of the *pwd*(1) command into the specified string space.

EXAMPLE

```
char *cwd, *getcwd();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
        perror("pwd");
        exit(1);
}
printf("%s\n", cwd);
```

SEE ALSO

malloc(3C), popen(3S), pwd(1).

RETURN VALUE

Returns NULL with *errno* set if *size* is not large enough, or if an error ocurrs in a lower-level function.

SUPPORT STATUS

Supported.

NAME

getenv − return value for environment name

SYNOPSIS

char *getenv (name)
char *name;

DESCRIPTION

Getenv searches the environment list (see *environ*(5)) for a string of the form *name=value*, and returns a pointer to the *value* in the current environment if such a string is present; if the *name* string is not found, *getenv* returns a NULL pointer.

SEE ALSO

exec(2), putenv(3C), environ(5).

SUPPORT STATUS

Supported.

NAME

getenv — return Fortran environment variable

SYNOPSIS

character*N c

call getenv(*env_var*, *c*)

DESCRIPTION

Getenv returns the character-string value of the environment variable represented by *env_var* into the character variable *c*. If no such environment variable exists, *getenv* returns all blanks.

SEE ALSO

getenv(3C), environ(5).

SUPPORT STATUS

Not supported.

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent — get group file entry

SYNOPSIS

#include <grp.h>

struct group *getgrent ( )

struct group *getgrgid (gid)
int gid;

struct group *getgrnam (name)
char *name;

void setgrent ( )

void endgrent ( )

struct group *fgetgrent (f)
FILE *f;

DESCRIPTION

Getgrent, *getgrgid* and *getgrnam* each return pointers to an object with the following structure containing the fields of a line in the /etc/group file. Each line contains a *group* structure, defined in the *<grp.h>* header file.

```
struct group {
        char     *gr_name;   /* the name of the group */
        char     *gr_passwd; /* the encrypted group

        int      gr_gid;     /* the numerical group ID */
        char     **gr_mem;   /* vector of pointers to member

};
```

getgrent

When first called, *getgrent* returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file. Therefore, successive calls to *getgrent* may be used to search the entire file.

getgrgid

Getgrgid searches from the beginning of the file until a numerical group id matching *gid* is found; *getgrgid* then returns a pointer to the particular structure in which the matching *gid* was found.

getgrnam

Getgrnam searches from the beginning of the file until a group name matching *name* is found; *getgrnam* then returns a pointer to the particular structure in which the matching *name* was found.

setgrent

A call to *setgrent* effectively rewinds the group file to allow repeated searches.

endgrent

Endgrent closes the group file.

fgetgrent

When called, *fgetgrent* returns a pointer to the next group structure in the stream *f* , which matches the format of **/etc/group.**

FILES

/etc/group

SEE ALSO

getlogin(3C), getpwent(3C), group(4).

RETURN VALUE

The *getgr* functions return a NULL pointer on EOF or error.

WARNING

The above routines use <stdio.h>. This causes them to increase the size of programs not otherwise using standard I/O more than might be expected.

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

getlogin — get login name

SYNOPSIS

char *getlogin ( );

DESCRIPTION

Getlogin returns a pointer to the login name found in /etc/utmp. *Getlogin* may be used in conjunction with *getpwnam* to locate the correct password file entry when the same user ID is shared by several login names.

If *getlogin* is called within a process that is not attached to a terminal, it returns a NULL pointer. The correct procedure for determining the login name is to call *cuserid*, or to call *getlogin* and if it fails to call *getpwuid*.

FILES

/etc/utmp

SEE ALSO

cuserid(3S), getgrent(3C), getpwent(3C), utmp(4).

RETURN VALUE

Getlogin returns the NULL pointer if *name* not found.

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

getopt − get option letter from argument vector

SYNOPSIS

int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;

DESCRIPTION

Getopt returns the next option letter in *argv* that matches a letter in *optstring*. *Optstring* is a string of recognized option letters; if a letter is followed by a colon, *getopt* expects the option to have an argument that may or may not be separated from it by white space. *Getopt* sets *optarg* to point to the start of the option argument.

Getopt places in *optind* the *argv* index of the next argument to be processed. Because *optind* is external, it is normally initialized to zero automatically before the first call to *getopt*.

When all options have been processed (i.e., up to the first non-option argument), *getopt* returns **EOF**. If the special option **−−** is used to delimit the end of the options, *getopt* returns **EOF** and skips **−−**.

EXAMPLE

The following code fragment processes the arguments for a command that can take the mutually exclusive options **a** and **b**, and the options **f** and **o**, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
        int c;
        extern int optind;
        extern char *optarg;
        :
        :
        while ((c = getopt (argc, argv, "abf:o:")) != EOF)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc( );
                        break;
```

```
                case 'f':
                        ifile = optarg;
                        break;
                case 'o':
                        ofile = optarg;
                        bufsiza = 512;
                        break;
                case '?':
                        errflg++;
                }
        if (errflg) {
                fprintf (stderr, "usage: . . . ");
                exit (2);
        }
        for ( ; optind < argc; optind++) {
                if (access (argv[optind], 4)) {
        .
        .
        .
}
```

SEE ALSO

getopt(1).

DIAGNOSTICS

Getopt prints an error message on *stderr* and returns a question mark (?) when it encounters an option letter not included in *optstring*.

WARNING

The above routine uses <stdio.h>, which increases the size of programs not otherwise using standard I/O more than might be expected.

SUPPORT STATUS

Supported.

NAME

getpass — read a password

SYNOPSIS

char *getpass (prompt)
char *prompt;

DESCRIPTION

Getpass reads up to a newline or EOF from the file /dev/tty, after prompting on the standard error output with the null-terminated string *prompt* and disabling echoing. *Getpass* returns a pointer to a null-terminated string of at most 8 characters. If /dev/tty cannot be opened, *getpass* returns a NULL pointer.

An interrupt terminates input and sends an interrupt signal to the calling program before returning.

FILES

/dev/tty

SEE ALSO

crypt(3C).

WARNING

The above routine uses <stdio.h>, which increases the size of programs not otherwise using standard I/O more than might be expected.

RESTRICTIONS

The return value points to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

getpw – get name from UID

SYNOPSIS

int getpw (uid, buf)
int uid;
char *buf;

DESCRIPTION

Getpw searches the password file for a user id number that equals *uid*, copies the line of the password file in which *uid* was found into the array pointed to by *buf*, and returns 0. *Getpw* returns non-zero if *uid* cannot be found.

This routine is included only for compatibility with prior UNIX systems and should not be used otherwise; see *getpwent*(3C) for routines to use instead.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), passwd(4).

RETURN VALUE

Getpw returns non-zero on error.

WARNING

The above routine uses **<stdio.h>**, which increases the size of programs not otherwise using standard I/O more than might be expected.

SUPPORT STATUS

Supported.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent — get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent ( )

struct passwd *getpwuid (uid)
int uid;

struct passwd *getpwnam (name)
char *name;

void setpwent ( )

void endpwent ( )


struct passwd *fgetpwent (f)
FILE *f;
```

DESCRIPTION

Getpwent, *getpwuid* and *getpwnam* each return a pointer to an object with the following structure containing the fields of a line in the /etc/passwd file. Each line in the file contains a *passwd* structure, declared in the *<pwd.h>* header file:

```
struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};

struct comment {
        char    *c_dept;
        char    *c_name;
        char    *c_acct;
        char    *c_bin;
};
```

This structure is declared in *<pwd.h>* so it is not necessary to redeclare it.

The *pw_comment* field is unused; the others have meanings described in *passwd*(4).

getpwent

When first called, *getpwent* returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file. Therefore, successive calls can be used to search the entire file.

getpwuid

Getpwuid searches from the beginning of the file until a numerical user id matching *uid* is found; *getpwuid* then returns a pointer to the particular structure in which the matching *uid* was found.

getpwnam

Getpwnam searches from the beginning of the file until a login name matching *name* is found; *getpwnam* then returns a pointer to the particular structure in which the matching *name* was found.

setpwent

A call to *setpwent* effectively rewinds the password file to allow repeated searches.

endpwent

Endpwent closes the password file.

fgetpwent

Fgetpwent returns a pointer to the next password structure in the stream *f* , which matches the format of **/etc/passwd.**

FILES

/etc/passwd

SEE ALSO

getlogin(3C), getgrent(3C), passwd(4).

RETURN VALUE

A NULL pointer is returned on EOF or error.

WARNING

The above routines use **<stdio.h>**, which increases the size of programs not otherwise using standard I/O more than might be expected.

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

gets, fgets — get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets (s)
char *s;

char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

DESCRIPTION

Gets reads characters from the standard input stream, *stdin*, into the array pointed to by *s*, until a new-line character is read or an end-of-file condition is encountered. *Gets* discards the new-line character and terminates the string with a null character.

Fgets reads characters from the *stream* into the array pointed to by *s*, until $n-1$ characters are read, or a new-line character is read and transferred to *s*, or an end-of-file condition is encountered. *Fgets* then terminates the string with a null character.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S).

RETURN VALUE

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned.

If a read error occurs (such as trying to read a file that has not been opened for reading) a NULL pointer is returned.

Otherwise *s* is returned.

SUPPORT STATUS

Supported.

NAME

getutent, getutid, getutline, pututline, setutent, endutent, utmpname — access utmp file entry

SYNOPSIS

```
#include <utmp.h>

struct utmp *getutent ( )

struct utmp *getutid (id)
struct utmp *id;

struct utmp *getutline (line)
struct utmp *line;

void pututline (utmp)
struct utmp *utmp;

void setutent ( )

void endutent ( )

void utmpname (file)
char *file;
```

DESCRIPTION

Getutent, *getutid* and *getutline* each return a pointer to a structure of the following type:

```
struct utmp {
        char      ut_user[8];        /* User login name */
        char      ut_id[4];          /* /etc/inittab id (usually line #)
                                     */
        char      ut_line[12];       /* device name (console, lnxx) */
        short     ut_pid;            /* process id */
        short     ut_type;           /* type of entry */
        struct    exit_status {
          short     e_termination;   /* Process termination status */
          short     e_exit;          /* Process exit status */
        } ut_exit;                   /* The exit status of a process
                                     marked as DEAD_PROCESS. */
        time_t    ut_time;           /* time entry was made */
};
```

getutent

Getutent reads in the next entry from a *utmp*-like file. If the file is not already open, *getutent* opens it. If it reaches the end of the file, *getutent* fails.

getutid

Getutid searches forward from the current point in the *utmp* file. If the type specified in *id* is RUN_LVL, BOOT_TIME, OLD_TIME or NEW_TIME, *getutid* returns a pointer to the first entry with a *ut_type* matching *id->ut_type*. If the type specified in *id* is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS or DEAD_PROCESS, then *getutid* returns a pointer to the first entry whose type is one of these four and whose *ut_id* field matches *id->ut_id*. If the end of file is reached without a match, *getutid* fails.

getutline

Getutline searches forward from the current point in the *utmp* file until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a *ut_line* string matching the *line->ut_line* string. If the end of file is reached without a match, *getutline* fails.

pututline

Pututline writes out the supplied *utmp* structure into the *utmp* file.

Pututline uses *getutid* to search forward for the proper place if it finds that it is not already at the proper place. Normally the user of *pututline* has already searched for the proper entry using one of the *getut* functions; in that case *pututline* does not search.

If *pututline* does not find a matching slot for the new entry, it adds a new entry to the end of the file.

setutent

Setutent resets the input stream to the beginning of the file. *Setutent* should be invoked before each search for a new entry if the user wishes to examine the entire file.

endutent

Endutent closes the currently open file.

utmpname

Utmpname changes the name of the file examined, from **/etc/utmp** to any other *file*, (**/etc/wtmp**). *Utmpname* does not return an error if *file* does not exist: *utmpname* does not open the file. It just closes the old file if it is currently open and saves the new file name.

FILES

/etc/utmp
/etc/wtmp

SEE ALSO

ttyslot(3C), utmp(4).

DIAGNOSTICS

A NULL pointer is returned upon failure to read (permissions or end of file condition) or upon failure to write.

COMMENTS

The most current entry is saved in a static structure. Multiple accesses require that the structure be copied before further accesses are made.

Each call to *getutid* or *getutline* examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, the function looks no further. Therefore, to use *getutline* to search for multiple occurences, the static structure must be zeroed out after each success, or *getutline* would just return the same pointer over and over again.

There is one exception to the rule about removing the structure before further reads: if *pututline* finds that it is not at the correct place in the file, the subsequent read does not hurt the contents of the static structure returned by the *getutent*, *getutid* or *getutline* routines, if the user has just modified those contents and passed the pointer back to *pututline*.

These routines use buffered standard I/O for input, but *pututline* uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the *utmp* and *wtmp* files.

SUPPORT STATUS

Supported.

NAME

hsearch, hcreate, hdestroy − manage hash search tables

SYNOPSIS

#include <search.h>

ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy ( )

DESCRIPTION

hsearch

Hsearch is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found.

Item is a structure of type ENTRY (defined in the <search.h> header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.)

Action is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made.

If *hsearch* does not resolve successfully, it returns a NULL pointer.

hcreate

Hcreate allocates sufficient space for the table, and must be called before *hsearch* is used.

Nel is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

hdestroy

Hdestroy destroys the search table, and may be followed by another call to *hcreate*.

NOTES

Hsearch uses *open addressing* with a *multiplicative* hash function. However, its source code has many other options available which the user may select by compiling the *hsearch* source with the following symbols defined to the preprocessor:

DIV Use the *remainder modulo table size* as the hash function instead of the multiplicative algorithm.

USCR Use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named *hcompar* and should behave in a mannner similar to *strcmp* (see *string*(3C)).

CHAINED Use a linked list to resolve collisions. If this option is selected, the following additional options become available:

START Place new entries at the beginning of the linked list (default is at the end).

SORTUP Keep the linked list sorted by key in ascending order.

SORTDOWN Keep the linked list sorted by key in descending order.

Additionally, preprocessor flags may be used to obtain debugging printout (**−DDEBUG**) and include a test driver in the calling routine (**−DDRIVER**). The source code should be consulted for further details.

EXAMPLE

The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
#include <stdio.h>
#include <search.h>

struct info {           /* this is the info stored in the */
        int age, room; /* table other than the key */
};
#define NUM_EMPL     5000     /* # of elements in */
                                 /* search table */

main( )
{
        /* space to store strings */
        char string_space[NUM_EMPL*20];
        /* space to store employee info */
        struct info info_space[NUM_EMPL];
        /* next avail space in string_space */
        char *str_ptr = string_space;
        /* next avail space in info_space */
        struct info *info_ptr = info_space;
        ENTRY item, *found_item, *hsearch( );
        /* name to look for in table */
        char name_to_find[30];
        int i = 0;

        /* create table */
        (void) hcreate(NUM_EMPL);
        while (scanf("%s%d%d", str_ptr, &info_ptr->age,
           &info_ptr->room) != EOF && i++ < NUM_EMPL) {
           /* put info in struct and struct in item */
          item.key = str_ptr;
          item.data = (char *)info_ptr;
          str_ptr += strlen(str_ptr) + 1;
```

```
                info_ptr++;
                /* put item into table */
                (void) hsearch(item, ENTER);
            }

            /* access table */
            item.key = name_to_find;
            while (scanf("%s", item.key) != EOF) {
                if ((found_item = hsearch(item, FIND)) != NULL) {
                    /* if item is in the table */
                    (void)printf("found %s, age = %d, room = %d\n",
                        found_item->key,
                        ((struct info *)found_item->data)->age,
                        ((struct info *)found_item->data)->room);
                } else {
                    (void)printf("no such employee %s\n",
                        name_to_find)
                }
            }
    }
```

SEE ALSO

bsearch(3C), lsearch(3C), malloc(3C), malloc(3X), string(3C), tsearch(3C).

DIAGNOSTICS

Hsearch returns a NULL pointer if:
  the action is **FIND** and the item could not be found
  the action is **ENTER** and the table is full

Hcreate returns zero if it cannot allocate sufficient space for the table.

WARNING

Hsearch and *hcreate* use *malloc*(3C) to allocate space.

RESTRICTIONS

Only one hash search table may be active at any given time.

SUPPORT STATUS

Supported.

NAME

hypot − Euclidean distance function

SYNOPSIS

#include <math.h>

double hypot (x, y)
double x, y;

DESCRIPTION

Hypot returns

sqrt(x * x + y * y),

taking precautions against unwarranted overflows.

DIAGNOSTICS

When the correct value would overflow, *hypot* returns **HUGE** and sets *errno* to **ERANGE.**

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M), sqrt(3F).

SUPPORT STATUS

Supported.

NAME

iargc − number of command line arguments

SYNOPSIS

integer i
i = iargc()

DESCRIPTION

The *iargc* function returns the number of command line arguments passed to the program. Thus, if a program were invoked by

```
prog arg1 arg2 arg3
```

iargc() would return 3.

SEE ALSO

getarg(3F).

SUPPORT STATUS

Not supported.

NAME
index — return location of Fortran substring

SYNOPSIS
character*N1 ch1
character*N2 ch2
integer i

i = index(ch1, ch2)

DESCRIPTION
Index returns the location of substring *ch2* in string *ch1*.

RETURN VALUE
positive-number
the position at which substring *ch2* starts

0 *ch2* is not in string *ch1*

SUPPORT STATUS
Not supported.

NAME

l3tol, ltol3 — convert between 3-byte integers and long integers

SYNOPSIS

```
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;

void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

DESCRIPTION

L3tol converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

Ltol3 performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

SEE ALSO

fs(4).

NOTE

The long integer is stored in four bytes, ordered high to low.

SUPPORT STATUS

Supported.

NAME

ldahread — read the archive header of a member of an archive file

SYNOPSIS

#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;

DESCRIPTION

If TYPE(*ldptr*) is the archive file magic number, *ldahread* reads the archive header of the common object file currently associated with *ldptr* into the area of memory beginning at *arhead*.

The program must be loaded with the object file access routine library **libld.a.**

RETURN VALUE

SUCCESS - successful completion
FAILURE - TYPE(*ldptr*) does not represent an archive file, or *ldahread* cannot read the archive header.

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldclose, ldaclose — close a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldclose (ldptr)
LDFILE *ldptr;

int ldaclose (ldptr)
LDFILE *ldptr;
```

DESCRIPTION

Ldopen(3X) and *ldclose* are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If TYPE(*ldptr*) does not represent an archive file, *ldclose* closes the file and frees the memory allocated to the LDFILE structure associated with *ldptr*. If TYPE(*ldptr*) is the magic number of an archive file, and if there are any more files in the archive, *ldclose* reinitializes OFFSET(*ldptr*) to the file address of the next archive member and returns FAILURE. The LDFILE structure is prepared for a subsequent *ldopen*(3X). In all other cases, *ldclose* returns SUCCESS.

Ldaclose closes the file and frees the memory allocated to the LDFILE structure associated with *ldptr* regardless of the value of TYPE(*ldptr*). *Ldaclose* always returns SUCCESS. The function is often used in conjunction with *ldaopen*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

fclose(3S), ldopen(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldfhread – read the file header of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

DESCRIPTION

Ldfhread reads the file header of the common object file currently associated with *ldptr* into the area of memory beginning at *filehead*.

In most cases the use of *ldfhread* can be avoided by using the macro **HEADER(*ldptr*)** defined in **ldfcn.h** (see*ldfcn*(4)). The information in any field, *fieldname*, of the file header may be accessed using **HEADER(*ldptr*).*fieldname***.

The program must be loaded with the object file access routine library **libld.a.**

RETURN VALUE

SUCCESS – successful completion
FAILURE – *ldfhread* could not read the file header

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldgetname — retrieve symbol name for common object file symbol table entry

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;

DESCRIPTION

Ldgetname returns a pointer to the name associated with **symbol** as a string. The string is contained in a static buffer local to *ldgetname* that is overwritten by each call to *ldgetname*, and therefore must be copied by the caller if the name is to be saved.

Ldgetname can be used to retrieve names from object files without any backward compatibility problems. *Ldgetname* returns NULL (defined in **stdio.h**) if the name cannot be retrieved. This situation can occur:

- if the string table cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to *ldgetname* that looks like a reference to a name in a non-existent string table), or
- if the name's offset into the string table is past the end of the string table.

Typically, *ldgetname* is called immediately after a successful call to *ldtbread* to retrieve the name associated with the symbol table entry filled by *ldtbread*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldlread, ldlinit, ldlitem — manipulate line number entries of a common object file function

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO linent;
```

DESCRIPTION

Ldlread searches the line number entries of the common object file currently associated with *ldptr*. *Ldlread* begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcnindx*, the index of its entry in the object file symbol table. *Ldlread* reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

Ldlinit and *ldlitem* together perform exactly the same function as *ldlread*. After an initial call to *ldlread* or *ldlinit*, *ldlitem* may be used to retrieve a series of line number entries associated with a single function. *Ldlinit* simply locates the line number entries for the function identified by *fcnindx*. *Ldlitem* finds and reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

Ldlread, *ldlinit*, and *ldlitem* each return either SUCCESS or FAILURE. *Ldlread* fails if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*. *Ldlinit* fails if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. *Ldlitem* fails if it finds no line number equal to or greater than *linenum*.

The programs must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbindex(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldlseek, ldnlseek — seek to line number entries of a section of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;

DESCRIPTION

Ldlseek seeks to the line number entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

Ldnlseek seeks to the line number entries of the section specified by *sectname*.

Ldlseek and *ldnlseek* return SUCCESS or FAILURE. *Ldlseek* fails if *sectindx* is greater than the number of sections in the object file; *ldnlseek* fails if there is no section name corresponding with **sectname*. Either function fails if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldohseek — seek to the optional file header of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;

DESCRIPTION

Ldohseek seeks to the optional file header of the common object file currently associated with *ldptr*.

The program must be loaded with the object file access routine library **libld.a**.

RETURN VALUE

SUCCESS - successful completion
FAILURE - the object file has no optional header
— *ldohseek* cannot seek to the optional header

SEE ALSO

ldclose(3X), ldopen(3X), ldfhread(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldopen, ldaopen — open a common object file for reading

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;

DESCRIPTION

Both *ldopen* and *ldaopen* open *filename* for reading. Both functions return NULL if *filename* cannot be opened, or if memory for the LDFILE structure cannot be allocated. A successful open does not insure that the given file is a common object file or an archived object file.

ldopen

Ldopen and *ldclose*(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If *ldptr* has the value NUll, then *ldopen* opens *filename*, allocates and initializes the LDFILE structure, and returns a pointer to the structure to the calling program.

If *ldptr* is valid and if TYPE(*ldptr*) is the archive magic number, *ldopen* reinitializes the LDFILE structure for the next archive member of *filename*.

Ldopen and *ldclose* are designed to work together. *Ldclose* returns FAILURE only when TYPE(*ldptr*) is the archive magic number and there is another file in the archive to be processed. Only then should *ldopen* be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, *ldopen* should be invoked with a NULL *ldptr* argument.

ldaopen

If the value of *oldptr* is not NULL, *ldaopen* opens *filename* anew and allocates and initializes a new LDFILE structure, copying the TYPE, OFFSET, and HEADER fields from *oldptr*.

Ldaopen returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, *oldptr*.

The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

The program must be loaded with the object file access routine library **libld.a**.

EXAMPLE

The following is a prototype for the use of *ldopen* and *ldclose*.

```
/* for each filename to be processed */

ldptr = NULL;
do
        if ( (ldptr = ldopen(filename, ldptr)) != NULL )

        {
                /* check magic number */
                /* process the file */
        }
} while (ldclose(ldptr) == FAILURE );
```

SEE ALSO

fopen(3S), ldclose(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldrseek, ldnrseek — seek to relocation entries of a section of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

Ldrseek seeks to the relocation entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

Ldnrseek seeks to the relocation entries of the section specified by *sectname*.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a.**

RETURN VALUE

SUCCESS - successful completion
FAILURE - *sectindx* exceeds the number of sections in the object file (*ldrseek*)
- no function name corresponds with *sectname* (*ldnrseek*)
- the specified section has no relocation entries (*ldrseek, ldnrseek*)
- the function cannot seek to the specified relocation entries (*ldrseek, ldnrseek*)

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldshread, ldnshread – read an indexed/named section header of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char sectname;
SCNHDR *secthead;
```

DESCRIPTION

Ldshread reads the section header specified by *sectindx* of the common object file currently associated with *ldptr* into the area of memory beginning at *secthead*.

Ldnshread reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

Note that the first section header has an index of *one*.

The program must be loaded with the object file access routine library libld.a.

RETURN VALUE

SUCCESS – successful completion
FAILURE – *sectindx* exceeds the number of sections in the object file (*ldshread*)
– there is no section name corresponding with *sectname* (*ldnshread*)
– the function cannot read the specified section header (*ldnshread, ldshread*)

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldsseek, ldnsseek − seek to an indexed/named section of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

Ldsseek seeks to the section specified by *sectindx* of the common object file currently associated with *ldptr*.

Ldnsseek seeks to the section specified by *sectname*.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a.**

RETURN VALUE

SUCCESS − successful completion
FAILURE − *sectindx* exceeds the number of sections in the object file (*ldsseek*)
− there is no section name corresponding with *sectname* (*ldnsseek*)
− there is no section data for the specified section (*ldsseek, ldnsseek*)
− the function cannot seek to the specified section (*ldsseek, ldnsseek*)

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldtbindex – compute index of symbol table entry of object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;

DESCRIPTION

Ldtbindex returns the **long** index of the symbol table entry at the current position of the common object file associated with *ldptr*.

The index returned by *ldtbindex* may be used in subsequent calls to *ldtbread*(3X). However, since *ldtbindex* returns the index of the symbol table entry that begins at the current position of the object file, if *ldtbindex* is called immediately after a particular symbol table entry has been read, it returns the index of the next entry.

Ldtbindex fails if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldtbread — read an indexed symbol table entry of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;

DESCRIPTION

Ldtbread reads the symbol table entry specified by *symindex* of the common object file currently associated with *ldptr* into the area of memory beginning at *symbol*.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

RETURN VALUE

SUCCESS — successful completion
FAILURE — *symindex* exceeds the number of symbols in the object file
— *ldtbread* cannot read the specified symbol table entry

SEE ALSO

ldclose(3X), ldopen(3X), ldtbseek(3X), ldgetname(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

ldtbseek — seek to the symbol table of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;

DESCRIPTION

Ldtbseek seeks to the symbol table of the object file currently associated with *ldptr*.

The program must be loaded with the object file access routine library **libld.a.**

RETURN VALUE

SUCCESS — successful completion
FAILURE — the symbol table has been stripped from the object file
— *ldtbseek* cannot seek to the symbol table

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldfcn(4).

SUPPORT STATUS

Supported.

NAME

len − return length of Fortran string

SYNOPSIS

character*N ch
integer i

i = len(ch)

DESCRIPTION

Len returns the length of string *ch*.

SUPPORT STATUS

Not supported.

NAME

lockf – record locking on files

SYNOPSIS

 # include <unistd.h>

lockf (fildes, function, size)
int fildes, function;
long size;

DESCRIPTION

The default locking routines, *locking* and *lockf*, are defined in the C library. See *lockf*(2) for more information on using these routines. This version of *lockf*(3X) resides in library *–llockf*, and is the standard UNIX version. Users who wish to use this version of *lockf* must link with the **lockf** library.

The *lockf* call allows sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file [see *chmod*(2)]. Calls to *lockf* from other processes which attempt to lock the locked file section either return an error value or put the calling process to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. [See *fcntl*(2)] for more information.]

Fildes is the file descriptor returned from a successful *open*, *creat*, *dup*, or *pipe* system call. The file descriptor must have write only (O_WRONLY) or read/write (O_RDWR) permission in order to establish lock with this function call.

Function is a control value which specifies the action to be taken. The permissible values for *function* are defined in <unistd.h> as follows:

```
#define F_ULOCK 0 /* Unlock a previously locked section */
#define F_LOCK  1 /* Lock a section for exclusive use */
#define F_TLOCK 2 /* Test and lock a section for exclusive use */
#define F_TEST  3 /* Test section for other processes locks */
```

F_UNLOCK removes locks from a section of the file. F_LOCK and F_TLOCK both lock a section of a file if the section is available. F_TEST is used to detect if a lock by another process is present on the specified section. All other function values are reserved for future extensions and result in an error return, if not implemented.

Size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is zero, the section from the current offset through the largest file offset (FCHAR_MAX) is locked; that is, from the current offset through the present or any future end-of-file. An area need not be allocated to the file in order to be locked, as such locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and F_TLOCK requests differ only by the action taken if the resource is not available. F_LOCK causes the calling process to sleep until the resource is available. F_TLOCK causes the function to return a −1 and set **errno** to an [EAGAIN] error if the section is already locked by another process.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to *lockf*or *fcntl* scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The *alarm*(2) call may be used to provide a timeout facility in applications which require this facility.

Both *lockf*(2) and *lockf*(3C) type locking calls can be issued concurrently, with the exception that all locking requests for a given file must be of the same type. An error exit is taken with *errno* set to EBUSY if a lock request is made to a file that already has existing locks of the other type.

While locks may be applied to special files or pipes, read/write operations are not blocked. Locks may not be applied to a directory.

FAILURE CONDITIONS

The *lockf* utility fails and sets *errno* if one or more of the following are true:

[EBADF]
: *Fildes* is not a valid open descriptor.

[EAGAIN]
: *Cmd* is F_TLOCK or F_TEST and the section is already locked by another process.

[EDEADLK]
: *Cmd* is F_LOCK or F_TLOCK and a deadlock would occur. Also the *cmd* is either of the above or F_ULOCK and the number of entries in the lock table would exceed the number allocated on the system.

[EBUSY]
Any *Cmd* issued against a file that already has locks of the wrong type. This refers to mixing *lockf*(2) locks and *lockf*(3C) locks on the same open file file.

WARNING

Note that *lockf*(2) and *lockf*(3C) calls, while being almost interface compatible are not functionally compatible unless the file permissions are set for mandatory lock enforcements on *write*(2) calls when using *lockf* (see *access*(2)). *Locking*(2) always enforces locks on *write*(2) call for regular files.

Unexpected results may occur in processes that do buffering in the user address space. The process may later read or write data which is or was locked. The standard I/O package is the most common source of unexpected buffering.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO

close(2), creat(2), fcntl(2), intro(2), lockf(2), open(2), read(2), write(2).

SUPPORT STATUS

Supported.

NAME

logname − return login name of user

SYNOPSIS

char *logname( )

DESCRIPTION

Logname returns a pointer to the null-terminated login name; it extracts the $LOGNAME variable from the environment of the user.

This routine is kept in **/usr/lib/libPW.a.**

FILES

/etc/profile

SEE ALSO

profile(4), environ(5), env(1), login(1).

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

This method of determining a login name is subject to forgery.

SUPPORT STATUS

Supported.

NAME

lsearch, lfind — linear search and update

SYNOPSIS

#include <stdio.h>
#include <search.h>

char *lsearch ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );

char *lfind ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );

DESCRIPTION

Lsearch is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, *lsearch* adds it at the end of the table.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

ARGUMENTS

Key points to the datum to be sought in the table.

Base points to the first element in the table.

Key and *base* should be of type pointer-to-element, and cast to type pointer-to-character.

Nelp points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table.

Compar is the name of the comparison function which the user must supply (*strcmp*, for example). *Compar* is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

This fragment reads in ≤ TABSIZE strings of length ≤ ELSIZE and stores them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

        char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
        unsigned nel = 0;
        int strcmp( );
        . . .
        while (fgets(line, ELSIZE, stdin) != NULL &&
           nel < TABSIZE)
              (void) lsearch(line, (char *)tab, &nel,
                       ELSIZE, strcmp);
        . . .
```

SEE ALSO

bsearch(3C), hsearch(3C), tsearch(3C).

DIAGNOSTICS

If the searched for datum is found, both *lsearch* and *lfind* return a pointer to it. Otherwise, *lfind* returns NULL and *lsearch* returns a pointer to the newly added element.

RESTRICTIONS

Undefined results can occur if there is not enough room in the table to add a new item.

SUPPORT STATUS

Supported.

NAME

malloc, free, realloc, calloc — main memory allocator

SYNOPSIS

char *malloc (size)
unsigned size;

void free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned nelem, elsize;

DESCRIPTION

malloc, free

Malloc and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to *free* is a pointer to a block previously allocated by *malloc*; after *free* is performed this space is made available for further allocation, but its contents are left undisturbed.

Undefined results will occur if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *sbrk* (see *brk*(2)) to get more memory from the system when there is no suitable space already free.

realloc

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, then *realloc* will ask *malloc* to enlarge the arena by *size* bytes and will then move the data to the new space.

Realloc also works if *ptr* points to a block freed since the last call of *malloc*, *realloc*, or *calloc*; thus sequences of *free*, *malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

calloc

Calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3X).

DIAGNOSTICS

Malloc, *realloc* and *calloc* return a NULL pointer if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When this happens the block pointed to by *ptr* may be destroyed.

NOTE

Search time increases when many objects have been allocated; that is, if a program allocates but never frees, then each successive allocation takes longer. For an alternate, more flexible implementation, see *malloc*(3X).

SUPPORT STATUS

Supported.

NAME

malloc, free, realloc, calloc, mallopt, mallinfo — fast main memory allocator

SYNOPSIS

#include <malloc.h>

char *malloc (size)
unsigned size;

void free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned nelem, elsize;

int mallopt (cmd, value)
int cmd, value;

struct mallinfo mallinfo (max)
int max;

DESCRIPTION

malloc, free

Malloc and *free* provide a simple general-purpose memory allocation package, which runs considerably faster than the *malloc*(3C) package. It is found in the library *malloc,* and is loaded if the option *−lmalloc* is used with *cc*(1) or *ld*(1).

Malloc returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to *free* is a pointer to a block previously allocated by *malloc*; after *free* is performed this space is made available for further allocation, and its contents have been destroyed (but see *mallopt* below for a way to change this behavior).

Undefined results will occur if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

realloc

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

calloc

Calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

mallopt

Mallopt provides for control over the allocation algorithm. The available values for *cmd* are:

M_MXFAST Set *maxfast* to *value*. The algorithm allocates all blocks below the size of *maxfast* in large groups and then doles them out very quickly. The default value for *maxfast* is 0.

M_NLBLKS — Set *numlblks* to *value*. The above mentioned large groups each contain *numlblks* blocks. *Numlblks* must be greater than 0. The default value for *numlblks* is 100.

M_GRAIN — Set *grain* to *value*. The sizes of all blocks smaller than *maxfast* are considered to be rounded up to the nearest multiple of *grain*. *Grain* must be greater than 0. The default value of *grain* is the smallest number of bytes which allows alignment of any data type. Value is rounded up to a multiple of the default when *grain* is set.

M_KEEP — Preserve data in a freed block until the next *malloc*, *realloc*, or *calloc*. This option is provided only for compatibility with the old version of *malloc* and is not recommended.

These values are defined in the <*malloc.h*> header file.

Mallopt may be called repeatedly, but may not be called after the first small block is allocated.

Mallinfo provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo {
        int arena;          /* total space in arena */
        int ordblks;        /* number of ordinary blocks */
        int smblks;         /* number of small blocks */
        int hblkhd;         /* space in holding block headers */
        int hblks;          /* number of holding blocks */
        int usmblks;        /* space in small blocks in use */
        int fsmblks;        /* space in free small blocks */
        int uordblks;       /* space in ordinary blocks in use */
        int fordblks;       /* space in free ordinary blocks */
        int keepcost;       /* space penalty if keep option */
                            /* is used */
}
```

This structure is defined in the <*malloc.h*> header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3C).

DIAGNOSTICS

Malloc, realloc and *calloc* return a NULL pointer if there is not enough available memory. When *realloc* returns NULL, the block pointed to by *ptr* is left intact. If *mallopt* is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

WARNINGS

This package usually uses more data space than *malloc*(3C).
The code size is also bigger than *malloc*(3C).

Note that unlike *malloc*(3C), this package does not preserve the contents of a block when it is freed, unless the M_KEEP option of *mallopt* is used.
Undocumented features of *malloc*(3C) have not been duplicated.

SUPPORT STATUS

Supported.

NAME

matherr – error-handling function

SYNOPSIS

#include <math.h>

int matherr (x)
struct exception *x;

DESCRIPTION

Matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors by including a function named *matherr* in their programs. *Matherr* must be of the form described above. A pointer to the exception structure *x* is passed to the user-supplied *matherr* function when an error occurs. This structure, which is defined in the *<math.h>* header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN	domain error
SING	singularity
OVERFLOW	overflow
UNDERFLOW	underflow
TLOSS	total loss of significance
PLOSS	partial loss of significance

The element *name* points to a string containing the name of the function that had the error. The variables *arg1* and *arg2* are the arguments to the function that had the error. *Retval* is a double precision number that is returned by the function having the error.

If the user-supplied *matherr* supplies a return value, that value must be non-zero. If the default error value is to be returned, the user-supplied *matherr* must return 0.

If *matherr* is not supplied by the user, the default error-handling procedures, described with the math functions involved, are invoked upon error. These procedures are summarized in the table below. In every case, *errno* is set to non-zero and the program continues.

EXAMPLE

```
matherr(x)
register struct exception *x;
{
   switch (x->type) {
   case DOMAIN:
   case SING: /* print message and abort */
      fprintf(stderr, "domain error in %s\n", x->name);
      abort( );
```

```
        case OVERFLOW:
            if (!strcmp("exp", x->name)) {
                /* if exp, print message, return the argument */
                fprintf(stderr, "exp of %f\n", x->arg1);
                x->retval = x->arg1;
            } else if (!strcmp("sinh", x->name)) {
                /* if sinh, set errno, return 0 */
                errno = ERANGE;
                x->retval = 0;
            } else
                /* otherwise, return HUGE */
                x->retval = HUGE;
            break;
        case UNDERFLOW:
            return (0); /* execute default procedure */
        case TLOSS:
        case PLOSS:
            /* print message and return 0 */
            fprintf(stderr, "loss of significance in %s\n", x->name);
            x->retval = 0;
            break;
        }
        return (1);
    }
```

DEFAULT ERROR HANDLING PROCEDURES						
	Types of Errors					
	DOMAIN	SING	OVER FLOW	UNDER FLOW	TLOSS	PLOSS
BESSEL: y0, y1, yn (neg. no.)	– M, –H	– –	H –	0 –	– –	* –
EXP:	–	–	H	0	–	
POW: (neg.)**(non-int.), 0**0	– M, 0	– –	H –	0 –	– –	– –
LOG: log(0): log(neg.):	 – M, –H	 M, –H –	 – –	 – –	 – –	 – –
SQRT:	M, 0	–	–	–	–	–
GAMMA:	–	M, H	–	–	–	–
HYPOT:	–	–	H	–	–	–
SINH, COSH:	–	–	H	–	–	–
SIN, COS:	–	–	–	–	M, 0	M, *
TAN:	–	–	H	–	0	*
ACOS, ASIN:	M, 0	–	–	–	–	–

ABBREVIATIONS	
*	As much as possible of the value is returned.
M	Message is printed.
H	HUGE is returned.
−H	−HUGE is returned.
0	0 is returned.

SUPPORT STATUS

Supported.

NAME

max, max0, amax0, max1, amax1, dmax1 – Fortran maximum-value functions

SYNOPSIS

integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l = max(i, j, k)
c = max(a, b)
d = max(a, b, c)
k = max0(i, j)
a = amax0(i, j, k)
i = max1(a, b)
d = amax1(a, b, c)
dp3 = dmax1(dp1, dp2)

DESCRIPTION

The maximum-value functions return the largest of their arguments of which there may be any number.

The generic form *max* can be used for all data types, and takes its return type from that of its arguments which must all be of the same type.

Max0 returns the integer form of the maximum value of its integer arguments; *amax0*, the real form of its integer arguments; *max1*, the integer form of its real arguments; *amax1*, the real form of its real arguments; and *dmax1*, the double-precision form of its double-precision arguments.

SEE ALSO

min(3F).

SUPPORT STATUS

Not supported.

NAME

mclock − return Fortran time accounting

SYNOPSIS

integer i

i = mclock( )

DESCRIPTION

Mclock returns time accounting information about the current process and its child processes. The value returned is the sum of the user time of the current process and the user and system times of all child processes.

SEE ALSO

times(2), clock(3C), system(3F).

SUPPORT STATUS

Not supported.

NAME

memccpy, memchr, memcmp, memcpy, memset — memory operations

SYNOPSIS

```
#include <memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy

Memccpy copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. *Memccpy* returns a pointer to the character after the copy of *c* in *s1*, or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

memchr

Memchr returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or a NULL pointer if *c* does not occur.

memcmp

Memcmp compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according to whether *s1* is lexicographically less than, equal to, or greater than *s2*.

Memcmp uses native character comparison, which is signed on the system.

memcpy

Memcpy copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

memset

Memset sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

NOTE

For user convenience, all these functions are declared in the optional *<memory.h>* header file.

RESTRICTIONS

Movement of characters between overlapping areas may yield unexpected results.

SUPPORT STATUS

Supported.

NAME

min, min0, amin0, min1, amin1, dmin1 — Fortran minimum-value functions

SYNOPSIS

integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l = min(i, j, k)
c = min(a, b)
d = min(a, b, c)
k = min0(i, j)
a = amin0(i, j, k)
i = min1(a, b)
d = amin1(a, b, c)
dp3 = dmin1(dp1, dp2)

DESCRIPTION

The minimum-value functions return the minimum of their arguments of which there may be any number.

Min is the generic form. *Min* can be used for all data types, and takes its return type from that of its arguments which must all be of the same type.

Min0 returns the integer form of the minimum value of its integer arguments; *amin0*, the real form of its integer arguments; *min1*, the integer form of its real arguments; *amin1*, the real form of its real arguments; and *dmin1*, the double-precision form of its double-precision arguments.

SEE ALSO

max(3F).

SUPPORT STATUS

Not supported.

NAME

mktemp – make a unique file name

SYNOPSIS

char *mktemp (template)
char *template;

DESCRIPTION

Mktemp replaces the contents of the string pointed to by *template* by a unique file name, and returns the address of *template*.

The string in *template* should be a file name with six trailing Xs. *Mktemp* replaces the Xs with a letter and the current process ID. *Mktemp* chooses the letter so that the resulting name does not duplicate an existing file.

SEE ALSO

getpid(2), tmpfile(3S), tmpnam(3S).

RESTRICTIONS

Mktemp may run out of letters.

SUPPORT STATUS

Supported.

NAME

nlist — get entries from name list

SYNOPSIS

#include <a.out.h>

int nlist (file-name, nl)
char *file-name;
struct nlist *nl[ ];

DESCRIPTION

Nlist examines the name list in the executable file whose name is pointed to by *file-name*, and selectively extracts a list of values and puts them in the array of nlist structures pointed to by *nl*.

The name list *nl* consists of an array of structures containing names of variables, types and values. The list is terminated with a null name; that is, a null string is in the name position of the structure.

Nlist looks up each variable name in the name list of the file. If the name is found, *nlist* inserts the type and value of the name in the next two fields. If the name is not found, *nlist* sets both entries to 0.

See *a.out*(4) for a discussion of the symbol table structure.

This subroutine is useful for examining the system name list kept in the file **/unix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(4).

RETURN VALUE

−1 Error; all type entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

0 Successful completion.

SUPPORT STATUS

Supported.

NAME

perror, errno, sys_errlist, sys_nerr — system error messages

SYNOPSIS

void perror (s)
char *s;

extern int errno;

extern char *sys_errlist[ ];

extern int sys_nerr;

DESCRIPTION

Perror produces a message on the standard error output, describing the last error encountered during a call to a system or library function.

The argument string *s* is printed first, then a colon and a blank, then the message and a new-line. To be of most use, the argument string should include the name of the function that incurred the error.

The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when successful calls are made.

To simplify variant formatting of messages, the array of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the new-line.

Sys_nerr is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2).

SUPPORT STATUS

Supported.

NAME

plot – graphics interface subroutines

SYNOPSIS

```
openpl ()

erase ()

label (s)
char *s;

line (x1, y1, x2, y2)
int x1, y1, x2, y2;

circle (x, y, r)
int x, y, r;

arc (x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

move (x, y)
int x, y;

cont (x, y)
int x, y;

point (x, y)
int x, y;

linemod (s)
char *s;

space (x0, y0, x1, y1)
int x0, y0, x1, y1;

closepl ()
```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. *Space* must be used before any of these functions to declare the amount of space necessary. See *plot*(4). *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

Circle draws a circle of radius *r* with center at the point *(x, y)*.

Arc draws an arc of a circle with center at the point *(x, y)* between the points *(x0, y0)* and *(x1, y1)*.

String arguments to *label* and *linemod* are terminated by nulls and do not contain new-lines.

See *plot*(4) for a description of the effect of the remaining functions.

The library files listed below provide several versions of these routines.

FILES

/usr/lib/libplot.a	produces output for *tplot*(1G) filters
/usr/lib/lib300.a	for DASI 300
/usr/lib/lib300s.a	for DASI 300s
/usr/lib/lib450.a	for DASI 450
/usr/lib/lib4014.a	for Tektronix 4014

WARNINGS

In order to compile a program containing these functions in *file.c* use

cc *file.c* −lplot

In order to execute the program, use

a.out | tplot

The above routines use **<stdio.h>**, which increases the size of programs not otherwise using standard I/O more than might be expected.

SEE ALSO

graph(1G), stat(1G), tplot(1G), plot(4).

SUPPORT STATUS

Not supported.

NAME

popen, pclose — initiate pipe to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen (command, type)
char *command, *type;

int pclose (stream)
FILE *stream;
```

DESCRIPTION

Popen creates a pipe between the calling program and the command to be executed. The arguments to *popen* are pointers to null-terminated strings. *Command* is a shell command line; *type* is an I/O mode, either **r** for reading or **w** for writing.

Popen returns a stream pointer. This allows the user to write to the standard input of the command, if the I/O mode is **w**, by writing to the file *stream*; or to read from the standard output of the command, if the I/O mode is **r**, by reading from the file *stream*.

A stream opened by *popen* should be closed by *pclose*. *Pclose* waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type **r** command may be used as an input filter and a type **w** as an output filter.

SEE ALSO

pipe(2), wait(2), fclose(3S), fopen(3S), system(3S).

RETURN VALUE

Popen returns a NULL pointer if files or processes cannot be created, or if the shell cannot be accessed.

Pclose returns −1 if *stream* is not associated with a *popen* command.

RESTRICTIONS

If the original and *popen* processes concurrently read or write a common file, neither should use buffered I/O, because the buffering becomes disordered. Problems with an output filter may be forestalled by careful buffer flushing, e.g. with *fflush*; see *fclose*(3S).

SUPPORT STATUS

Supported.

NAME

printf, fprintf, sprintf — print formatted output

SYNOPSIS

#include <stdio.h>

int printf (format [, arg] ...)
char *format;

int fprintf (stream, format [, arg] ...)
FILE *stream;
char *format;

int sprintf (s, format [, arg] ...)
char *s, format;

DESCRIPTION

Printf writes it *args* on the standard output stream *stdout*.

Fprintf write its *args* on the named output *stream*.

Sprintf write its *args*, followed by the null character (**\0**), into consecutive bytes starting at **s*; it is the responsibility of the user to ensure that enough storage is available.

Each function returns the number of characters transmitted (not including the **\0** in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more *arg*s. The results of formatting are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the function ignores the excess *arg*s.

CONVERSION SPECIFICATIONS

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

> Zero or more *flag* characters, which modify the meaning of the conversion specification.
>
> An optional decimal digit string, which specifies a minimum *field width*. If the converted value has fewer characters than the field width, the field is padded on the left (or right, if the left-adjustment flag character has been given) to the field width.
>
> A *precision*, which specifies the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision is a period (**.**) followed by a decimal digit string: a null digit string is treated as zero.

An optional **l**, which specifies that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putc*(3S) had been called.

Flag characters

– Left justify the result of the conversion within the field.

+ Precede the result of a signed conversion with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, prefix a blank to the result. If the blank and + flags both appear, the blank flag is ignored.

Convert the value to an *alternate form*. For **c**, **d**, **s**, and **u** conversions, the flag has no effect.

For **o** conversion, increase the precision to force the first digit of the result to be a zero.

For x (X) conversion, prefix a non-zero result with **0x** (**0X**). prefixed to it.

For e, **E**, **f**, **g**, and **G** conversions, force the result to contain a decimal point, even if no digits follow the point. Normally, a decimal point appears in the result of these conversions only if a digit follows it.

For **g** and **G** conversions, do *not* remove trailing zeroes from the result. Trailing zeros are normally removed.

Conversion characters

d,o,u,x,X Convert the integer *arg* to signed decimal, unsigned octal, decimal, or hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion.

The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, the value is expanded with leading zeroes. The default precision is 1.

The result of converting a zero value with a precision of zero is a null string.

f Convert the float or double *arg* to decimal notation in the style [−]*ddd.ddd*, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears.

e,E Convert the float or double *arg* in the style [−]*d.ddd*e±*dd*, where there is one digit before the decimal point and the number of digits after it is equal to the precision. If the precision is missing, produce 6 digits; if the precision is zero, eliminate the decimal point.

The **E** format code produces a number with **E** instead of **e** introducing the exponent.

The exponent always contains at least two digits.

g,G Print the float or double *arg* in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** is used only if the exponent resulting from the conversion is less than −4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

c Print the character *arg*.

s Assume the *arg* to be a string (character pointer); print characters from the string until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed.

If the string pointer *arg* has the value zero, the result is undefined.

A *null* arg yields undefined results.

% Print a %; no argument is converted.

EXAMPLES

To print a date and time in the form:

Sunday, July 3, 10:02

where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", weekday, month, day, hour, min);
```

To print π to five decimal places:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

ecvt(3C), putc(3S), scanf(3S), stdio(3S).

SUPPORT STATUS

Supported.

NAME

putc, putchar, fputc, putw — put character or word on a stream

SYNOPSIS

#include <stdio.h>

int putc (c, stream)
int c;
FILE *stream;

int putchar (c)
int c;

int fputc (c, stream)
int c;
FILE *stream;

int putw (w, stream)
int w;
FILE *stream;

DESCRIPTION

Putc writes the character *c* onto the output *stream* at the position where the file pointer, if defined, is pointing. *Putchar*(*c*) is defined as *putc*(*c, stdout*). *Putc* and *putchar* are macros.

Fputc behaves like *putc*, but is a function rather than a macro. *Fputc* runs more slowly than *putc*, but uses less space per invocation.

Putw writes the word *w* to the output *stream* at the position at which the file pointer, if defined, is pointing. The size of a word is the size of an integer. *Putw* neither assumes nor causes special alignment in the file.

The output streams to which these functions write may be *buffered* or *unbuffered*. When an output stream is *unbuffered* information is queued for writing on the destination file or terminal as soon as written. When it is *buffered* many characters are saved up and written as a block. When it is *line-buffered*, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested).

By default, output streams are *buffered* if the output refers to a file, and *line-buffered* if the output refers to a terminal. The standard error output stream *stderr* is the only exception: by default *stderr* is *unbuffered*.

The *freopen* command (see *fopen*(3S)) may be used to alter the buffering of *stderr* to *buffered* or *line-buffered*. *Setbuf*(3S) may be used to alter the buffering of other output streams.

SEE ALSO

fclose(3S), ferror(3S), fopen(3S), fread(3S), printf(3S), puts(3S), setbuf(3S).

RETURN VALUE

Success: the value the function has written

Failure: the constant **EOF**, because the stream is not open for writing, or the output file cannot be expanded

Note that because **EOF** is a valid integer, *ferror*(3S) should be used to detect *putw* errors.

RESTRICTIONS

Because it is implemented as a macro, *putc* incorrectly handles a *stream* argument with side effects. In particular, **putc(c, *f++);** does not work correctly. *Fputc* should be used instead.
Because of possible differences in word length and byte ordering, files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor. For this reason the use of *putw* should be avoided.

SUPPORT STATUS

Supported.

NAME

putenv — change or add value to environment

SYNOPSIS

int putenv (string)
char *string;

DESCRIPTION

String points to a string of the form *name=value. Putenv* makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* becomes part of the environment, so altering the string changes the environment. The space used by *string* is no longer used once a new string-defining *name* is passed to *putenv*.

DIAGNOSTICS

Putenv returns non-zero if it was unable to obtain enough space via *malloc* for an expanded environment, otherwise zero.

SEE ALSO

exec(2), getenv(3C), malloc(3C), environ(5).

WARNINGS

Putenv manipulates the environment pointed to by *environ*, and can be used in conjunction with *getenv*. However, *envp* (the third argument to *main*) is not changed.

This routine uses *malloc*(3C) to enlarge the environment.

After *putenv* is called, environmental variables are not in alphabetical order.

A potential error is to call *putenv* with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

SUPPORT STATUS

Supported.

NAME

putpwent — write password file entry

SYNOPSIS

#include <pwd.h>

```
int putpwent (p, f)
struct passwd *p;
FILE *f;
```

DESCRIPTION

Putpwent is the inverse of *getpwent*(3C). Given a pointer to a *passwd* structure created by *getpwent* (or *getpwuid* or *getpwnam*), *putpwent* writes a line on the stream *f* which matches the format of /etc/**passwd.**

RETURN VALUE

non-zero	error
0	successful completion

SEE ALSO

getpwent(3C).

WARNING

The above routine uses <**stdio.h**>, which increases the size of programs not otherwise using standard I/O more than might be expected.

SUPPORT STATUS

Supported.

NAME

puts, fputs — put a string on a stream

SYNOPSIS

```
#include <stdio.h>

int puts (s)
char *s;

int fputs (s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Puts writes the null-terminated string pointed to by *s*, followed by a new-line character, to the standard output stream *stdout*.

Fputs writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminating null character.

Note that *puts* appends a new-line character, but *fputs* does not.

DIAGNOSTICS

Both routines return NULL on error. This occurs if the routines try to write on a file that has not been opened for writing.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), printf(3S), putc(3S).

SUPPORT STATUS

Supported.

NAME

qsort — quicker sort

SYNOPSIS

void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned int nel;
int (*compar)( );

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

ARGUMENTS

Base points to the element at the base of the table. This pointer should be of type pointer-to-element, and cast to type pointer-to-character.

Nel is the number of elements in the table.

Compar is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

SEE ALSO

bsearch(3C), lsearch(3C), string(3C), sort(1).

SUPPORT STATUS

Supported.

NAME

rand, srand — simple random-number generator

SYNOPSIS

int rand ( )

void srand (seed)
unsigned seed;

DESCRIPTION

Rand uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

Srand can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTE

The spectral properties of *rand* leave a great deal to be desired. *Drand48*(3C) provides a much better, though more elaborate, random-number generator.

SEE ALSO

drand48(3C).

SUPPORT STATUS

Supported.

NAME

rand, srand, irand — Fortran uniform random-number generator

SYNOPSIS

integer i, j

call srand(iseed)
i = irand( )
j = rand( )

DESCRIPTION

Irand generates succussive pseudo-random numbers in the range from 0 to 2**15-1. *Rand* generates pseudo-random numbers distributed in (0, 1.0). *Srand* takes its integer argument as the seed of a random-number generator, the values of which are returned through successive invocations of *rand*.

SEE ALSO

rand(3C).

SUPPORT STATUS

Not supported.

NAME

regcmp, regex — compile and execute regular expression

SYNOPSIS

char *regcmp(string1 [, string2, ...], 0)
char *string1, *string2, ...;

char *regex(re, subject[, ret0, ...])
char *re, *subject, *ret0, ...;

extern char *loc1;

DESCRIPTION

regcmp

Regcmp compiles a regular expression and returns a pointer to the compiled form. *Regcmp* uses *malloc*(3C) to create space for the vector. It is the responsibility of the user to free unneeded space so allocated.

A NULL return from *regcmp* indicates an incorrect argument.

Regcmp(1) has been written to generally preclude the need for this routine at execution time.

regex

Regex executes a compiled pattern against the subject string. The additional arguments contain the matched pattern(s) after *regex* is called. *Regex* returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer *loc1* points to where the match began.

Regcmp and *regex* work similarly to the editor, *ed*(1), however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

[ ] * . ^ These symbols retain their current meaning.

$ This symbol matches the end of the string, **\n** matches the new-line.

— Within brackets the minus means *through*. For example, [a—z] is equivalent to [abcd...xyz]. The — matches itself only if used as the last or first character. For example, the character class expression []—] matches the characters] and —.

+ A regular expression followed by + means *one or more times*. For example, [0—9]+ is equivalent to [0—9][0—9]*.

{m} {m,} {m,u}
Integer values enclosed in { } indicate the number of times the preceding regular expression is to be applied. *m* is the minimum number and *u* is a number, less than 256, which is the maximum. If only *m* is present (e.g., {m}), *m* indicates the exact number of times the regular expression is to be applied. {m,} is analogous to {m,infinity}. The plus (+) and asterisk (*) operations are equivalent to {1,} and {0,} respectively.

(...)$*n*
The value of the enclosed regular expression is to be

returned in the *(n+1)*th argument following the subject argument. At most ten enclosed regular expressions are allowed. *Regex* makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g. *, +, { }, can work on a single character or a regular expression enclosed in parenthesis. For example, (a*(cb+)*)$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

This routine is kept in **/usr/lib/libPW.a.**

EXAMPLES

Example 1:

This example matches a leading new-line in the subject string pointed at by cursor.

```
char *cursor, *newcursor, *ptr;
        ...
newcursor = regex((ptr = regcmp("^\n", 0)), cursor);
free(ptr);
```

Example 2:

This example matches through the string **Testing3** and returns the address of the character after the last matched character (cursor+11). The string **Testing3** is copied to the character array *ret0*.

```
char ret0[9];
char *newcursor, *name;
        ...
name = regcmp("([A−Za−z][A−za−z0−9_]{0,7})$0", 0);
newcursor = regex(name, "123Testing321", ret0);
```

Example 3:

This example applies a precompiled regular expression in file.i (see *regcmp*(1)) against *string*.

```
#include "file.i"
char *string, *newcursor;
        ...
newcursor = regex(name, string);
```

SEE ALSO

malloc(3C), ed(1), regcmp(1).

RESTRICTIONS

The user program may run out of memory if *regcmp* is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for *malloc*(3C) reuses the same vector saving time and space:

```
/* user program */
        ...
malloc(n) {
        static int rebuf[256];
        return rebuf;
}
```

SUPPORT STATUS

Supported.

NAME

anint, dnint, nint, idnint — Fortran nearest integer functions

SYNOPSIS

integer i
real rl, r2
double precision dpl, dp2

r2 = anint(rl)
i = nint(rl)

dp2 = anint(dpl)
dp2 = dnint(dpl)

i = nint(dpl)
i = idnint(dpl)

DESCRIPTION

Anint returns the nearest whole real number to its real argument *r1*. This whole number is the nearest integer greater than *r1* if *r1* is greater than zero, and the nearest integer less than *r1* if *r1* if less than zero. *Dnint* does the same for its double-precision argument.

Nint returns the nearest integer to its real argument. *Idnint* is the double-precision version.

Anint, the generic form of *anint* and *dnint*, returns the data type of its argument. *Nint*, the generic form of *idnint*, returns the nearest whole number to its argument; the type of the returned value is the same as the argument.

SUPPORT STATUS

Not supported.

NAME

scanf, fscanf, sscanf — convert formatted input

SYNOPSIS

#include <stdio.h>

int scanf (format [, pointer] ... **)**
char *format;

int fscanf (stream, format [, pointer] ... **)**
FILE *stream;
char *format;

int sscanf (s, format [, pointer] ... **)**
char *s, *format;

DESCRIPTION

Scanf reads from the standard input stream *stdin*.

Fscanf reads from the named input *stream*.

Sscanf reads from the character string *s*.

Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored.

Scanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

Scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

CONVERSION SPECIFICATIONS

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, an optional **l** or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. The suppression of assignment effectively skips the described input field. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument should be given. The following conversion codes are allowed:

% a single % is expected in the input at this point; no assignment is made.

The conversion characters **d**, **u**, **o**, and **x** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list.

d a decimal integer is expected; the corresponding argument should be an integer pointer.

u an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

o an octal integer is expected; the corresponding argument should be an integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

e,f,g a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*.

The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an E or an e, followed by an optionally signed integer.

The conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which is added automatically. The input field is terminated by a white-space character.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use %1s. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

[string data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, called the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex, (**^**), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string.

A range of characters may be represented by the construct *first-last*, thus [0123456789] may be expressed [0-9]. If *first* is lexically greater than *last*, however, the dash stands for itself. The dash also stands for itself whenever it is the first or the last character in the scanset.

To include the right square bracket as an element of the scanset, it must be the first character (possibly preceded by a circumflex) of the scanset; in this case it is not syntactically interpreted as the closing bracket.

The corresponding argument of the *scanf* system call must point to a character array large enough to hold the data field and the terminating \0, which is added automatically.

EXAMPLES

Example 1

```
int i; float x; char name[50];
scanf ("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E−1 thompson
```

assigns **25** to *i*, **5.432** to *x*, and places **thompson\0** in *name*.

Example 2

```
int i; float x; char name[50];
scanf ("%2d%f%*d %[0-9]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns **56** to *i*, **789.0** to *x*, skips **0123**, and places the string **56\0** in *name*. The next call to *getchar* (see *getc*(3S)) returns **a**.

SEE ALSO

atof(3C), getc(3S), printf(3S), strtol(3C).

NOTE

Trailing white space (including a new-line) is left unread unless matched in the control string.

DIAGNOSTICS

These functions return **EOF** on end of input and a short count for missing or invalid data items.

RESTRICTIONS

The success of literal matches and suppressed assignments cannot be directly determined.

SUPPORT STATUS

Supported.

NAME

setbuf — assign buffering to a stream

SYNOPSIS

#include <stdio.h>

void setbuf (stream, buf)
FILE *stream;
char *buf;

DESCRIPTION

Setbuf alters the buffering of *stream*. *Setbuf* is used after a stream has been opened but before it is read or written.

If *buf* points to a character array, that array is used as the buffer instead of the automatically allocated buffer. If *buf* is a NULL character pointer, *setbuf* completely removes input/output buffering for *stream*.

A constant **BUFSIZ**, defined in the **<stdio.h>** header file, contains the *size* of array needed to buffer stream:

    char buf[BUFSIZ];

A buffer is normally obtained from *malloc*(3C) at the time of the first *getc* or *putc*(3S) on the file. The standard error stream *stderr* is normally unbuffered.

Output streams directed to terminals are always line-buffered unless they are unbuffered using *setbuf.*

SEE ALSO

fopen(3S), getc(3S), malloc(3C), putc(3S).

NOTE

A common source of error is allocating buffer space as an *automatic* variable in a code block, and then failing to close the stream in the same block.

SUPPORT STATUS

Supported.

NAME

setjmp, longjmp — non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp (env)
jmp_buf env;

void longjmp (env, val)
jmp_buf env;
int val;
```

DESCRIPTION

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp*. The type of *env*, *jmp_buf*, is defined in the *<setjmp.h>* header file. *Setjmp* returns the value 0.

Longjmp restores the environment saved by the last call of *setjmp* with the corresponding *env* argument. After *longjmp* is completed program execution continues as if the corresponding call of *setjmp* had just returned the value *val*.

Longjmp cannot cause *setjmp* to return the value 0. If *longjmp* is invoked with a second argument of 0, *setjmp* returns 1. All accessible data have values as of the time *longjmp* was called.

SEE ALSO

signal(2).

WARNING

If *longjmp* is called when *env* was not primed by a call to *setjmp*, or when the last such call is in a function which has since returned, absolute chaos is guaranteed.

SUPPORT STATUS

Supported.

NAME

sign, isign, dsign — Fortran transfer-of-sign intrinsic function

SYNOPSIS

integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k = isign(i, j)
k = sign(i, j)

r3 = sign(r1, r2)

dp3 = dsign(dp1, dp2)
dp3 = sign(dp1, dp2)

DESCRIPTION

Isign returns the magnitude of its first argument with the sign of its second argument. *Sign* and *dsign* are its real and double-precision counterparts, respectively.

The generic version, *sign* returns the manitude of the first argument with the sign of its second argument; the return value has the same type as the arguments.

SUPPORT STATUS

Not supported.

NAME
signal – specify Fortran action on receipt of a system signal

SYNOPSIS
integer i
external integer intfnc

call signal(i, intfnc)

DESCRIPTION
Signal specifies a function to be invoked when the current process receives a specific signal. The first argument specifies the fault or exception. The second argument specifies the function to be invoked.

SEE ALSO
kill(2), signal(2).

SUPPORT STATUS
Not supported.

NAME

sleep — suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION

Sleep suspends the current process from execution for the number of *seconds* specified by the argument.

The actual suspension time may be less than that requested for two reasons:

—scheduled wakeups occur at fixed 1-second intervals, on the second, according to an internal clock
—any caught signal terminates the *sleep* following execution of the catching routine of that signal.

Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system.

Sleep returns the the requested sleep time minus the time actually slept, in case the caller had requested another alarm to occur earlier than the end of the requested *sleep* time, or the *sleep* was interrupted due to another caught signal.

Sleep is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. *Sleep* saves and restores the previous state of the alarm signal.

The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time until such alarm signal, the process sleeps only until the alarm signal would have occurred, and the alarm catch routine of the caller is executed just before the *sleep* routine returns; but if the *sleep* time is less than the time until such alarm, *sleep* resets the prior alarm time to go off at the same time it would have without the intervening *sleep*.

SEE ALSO

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

RESTRICTIONS

Because of the 1 second granularity of the timer, it is possible for the alarm to expire before it is waited for when *seconds* is small (such as 1). This could result in the process sleeping forever.

SUPPORT STATUS

Supported.

NAME

sputl, sgetl — access long numeric data in a machine independent fashion.

SYNOPSIS

sputl (value, buffer)
long value;
char *buffer;

long sgetl (buffer)
char *buffer;

DESCRIPTION

Sputl(3X) takes the 4 bytes of the long *value* and places them in memory starting at the address pointed to by *buffer*. The ordering of the bytes is the same on all machines.

Sgetl retrieves the 4 bytes in memory starting at the address pointed to by *buffer* and returns the long value in the byte ordering of the host machine.

The usage of *sputl*(3X) and *sgetl* in combination provides a machine independent way of storing long numeric data in an ASCII file. The numeric data stored in the portable archive file format (see *ar*(4)) is moved in and out of buffers with *sputl*(3X) and *sgetl* respectively.

A program which uses these functions must be loaded with the object file access routine library **libld.a**.

SEE ALSO

ar(4).

SUPPORT STATUS

Supported.

NAME

ssignal, gsignal – software signals

SYNOPSIS

#include <signal.h>

int (*ssignal (sig, action))( )
int sig, (*action)( );

int gsignal (sig)
int sig;

DESCRIPTION

Ssignal and *gsignal* implement a software facility similar to *signal*(2). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and is also available to users for their own purposes.

ssignal

Software signals available to users are associated with integers in the inclusive range 1 through 15. A call to *ssignal* associates a procedure, *action*, with the software signal *sig*.

The first argument to *ssignal* is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a user defined *action function* or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). *Ssignal* returns the action previously established for that signal type; if no action has been established or the signal number is invalid, *ssignal* returns SIG_DFL.

gsignal

Gsignal raises the signal identified by its argument, *sig*:

> If an action function has been established for *sig*, then *gsignal* resets that action to SIG_DFL and enters the action function associated with argument *sig*. *Gsignal* returns the value returned to it by the action function.
>
> If the action for *sig* is SIG_IGN, *gsignal* returns the value 1 and takes no other action.
>
> If the action for *sig* is SIG_DFL, *gsignal* returns the value 0 and takes no other action.
>
> If *sig* has an invalid value or no action was specified for *sig*, *gsignal* returns the value 0 and takes no other action.

SEE ALSO

signal(2).

NOTES

There are some additional signals with numbers outside the range 1 through 15 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are valid, although their use may interfere with the operation of the Standard C Library.

SUPPORT STATUS

Supported.

NAME

stdio − standard buffered input/output package

SYNOPSIS

#include <stdio.h>

FILE *stdin, *stdout, *stderr;

DESCRIPTION

The functions described in the entries of sub-class 3S of this manual constitute an efficient, user-level I/O buffering scheme. The in-line macros *getc*(3S) and *putc*(3S) handle characters quickly. The macros *getchar*, *putchar*, and the higher-level routines *fgetc*, *fgets*, *fprintf*, *fputc*, *fputs*, *fread*, *fscanf*, *fwrite*, *gets*, *getw*, *printf*, *puts*, *putw*, and *scanf* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. *Fopen*(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions.

Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file.

A constant NULL (0) designates a nonexistent pointer.

RETURN VALUE

An integer constant **EOF** (−1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

FILES

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

#include <stdio.h>

The functions and constants in the entries of sub-class 3S of this manual are declared in that header file and need no further declaration.

WARNING

The constants and the following *functions* are implemented as macros (Redeclaration of these names is perilous): *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, *clearerr*, and *fileno*.

SEE ALSO

open(2), close(2), lseek(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S).

DIAGNOSTICS

Invalid *stream* pointers usually cause serious errors, possibly including program termination. Individual function descriptions describe the possible error conditions.

SUPPORT STATUS

Supported.

NAME

stdipc — standard interprocess communication package

SYNOPSIS

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

```
key_t ftok(path, id)
char *path;
char id;
```

DESCRIPTION

Ftok returns a key based on *path* and *id* that is usable in subsequent *msgget*, *semget* and *shmget* system calls. *Path* must be the path name of an existing file that is accessible to the process. *Id* is a character which uniquely identifies a project.

Ftok returns the *same* key for linked files when called with the same *id* and it returns *different* keys when called with the same file name but different *ids*.

NOTE

All interprocess communication facilities require the user to supply a key to be used by the *msgget*(2), *semget*(2) and *shmget*(2) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is the *ftok* subroutine described above. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number.

There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it becomes possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

SEE ALSO

intro(2), msgget(2), semget(2), shmget(2).

DIAGNOSTICS

Ftok returns (key_t) −1 if *path* does not exist or if it is not accessible to the process.

WARNING

If the file whose *path* is passed to *ftok* is removed when keys still refer to the file, future calls to *ftok* with the same *path* and *id* return an error. If the same file is recreated, then *ftok* is likely to return a different key than it did the original time it was called.

SUPPORT STATUS

Supported.

NAME

lge, lgt, lle, llt – string comparision intrinsic functions

SYNOPSIS

character*N a1, a2
logical l

l = lge (a1,a2)
l = lgt (a1,a2)
l = lle (a1,a2)
l = llt (a1,a2)

DESCRIPTION

These functions return .TRUE. if the inequality holds and .FALSE. otherwise.

SUPPORT STATUS

Not supported.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok — string operations

SYNOPSIS

```
#include <string.h>

char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s, c;

char *strrchr (s, c)
char *s, c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;
```

DESCRIPTION

The arguments *s1, s2* and *s* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy* and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*. *Strncmp* makes the same comparison but looks at at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result is not null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

Strlen returns the number of characters in *s*, not including the terminating null character.

Strchr (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered part of the string.

Strpbrk returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

Strspn (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

Strtok considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call to *strtok* (with pointer *s1* specified) returns a pointer to the first character of the first token, and writes a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) work through the string *s1* immediately following that token. In this way subsequent calls work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, *strtok* returns NULL pointer.

NOTE

For user convenience, these functions are all declared in the optional <*string.h*> header file.

RESTRICTIONS

Strcmp and *strncmp* use native character comparison, which is signed on the system.

Movement of characters between overlapping areas may yield unexpected results.

SUPPORT STATUS

Supported.

NAME

strtod, atof — convert string to double-precision number

SYNOPSIS

double strtod (str, ptr)
char *str, **ptr;

double atof (str)
char *str;

DESCRIPTION

Strtod returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character.

Strtod recognizes an optional string of white-space characters (as defined by *isspace* in *ctype*(3C)), then an optional sign, then a string of digits optionally containing a decimal point, then an optional **e** or **E** followed by an optional sign or space, followed by an integer.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, **ptr* is set to *str*, and zero is returned.

Atof(str) is equivalent to *strtod(str, (char **)NULL)*.

SEE ALSO

ctype(3C), scanf(3S), strtol(3C).

DIAGNOSTICS

If the correct value would cause overflow, ±HUGE is returned (according to the sign of the value), and *errno* is set to ERANGE.
If the correct value would cause underflow, zero is returned and *errno* is set to ERANGE.

SUPPORT STATUS

Supported.

NAME

strtol, atol, atoi — convert string to integer

SYNOPSIS

long strtol (str, ptr, base)
char *str;
char **ptr;
int base;

long atol (str)
char *str;

int atoi (str)
char *str;

DESCRIPTION

Strtol returns the value represented by the character string *str* as a long integer. *Strtol* scans the string up to the first character inconsistent with the base, ignoring leading white-space characters.

If the value of *ptr* is not (char **)NULL, *strtol* returns a pointer to the character terminating the scan in **ptr*. If no integer can be formed, *strtol* sets **ptr* to *str*, and returns zero.

If *base* is positive (and not greater than 36), *strtol* uses *base* as the base for conversion. After an optional leading sign on *str*, *strtol* ignores leading zeros, as well as 0x and 0X if *base* is 16.

If *base* is zero, the string itself determines the base: after an optional leading sign, a leading zero indicates octal conversion, and a leading 0x or 0X indicates hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from *long* to *int* can, of course, take place upon assignment, or by an explicit cast.

Atol(str) is equivalent to *strtol(str, (char **)NULL, 10)*.

Atoi(str) is equivalent to *(int) strtol(str, (char **)NULL, 10)*.

SEE ALSO

ctype(3C), scanf(3S), strtod(3C).

RESTRICTIONS

Overflow conditions are ignored.

SUPPORT STATUS

Supported.

NAME

swab – swap bytes

SYNOPSIS

void swab (from, to, nbytes)
char *from, *to;
int nbytes;

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the array pointed to by *to*, exchanging adjacent even and odd bytes. *Nbytes* should be even and non-negative. If *nbytes* is odd and positive *swab* uses *nbytes*−1 instead. If *nbytes* is negative *swab* does nothing.

Swab is useful for carrying binary data between machines.

SUPPORT STATUS

Supported.

NAME
system — issue a shell command from Fortran

SYNOPSIS
character*N c

call system(c)

DESCRIPTION
System passes its character argument to *sh*(1) as input, as if the string had been typed at a terminal. The current process waits until the shell has completed.

SEE ALSO
exec(2), system(3S), sh(1).

SUPPORT STATUS
Not supported.

NAME

system − issue a shell command

SYNOPSIS

#include <stdio.h>

int system (string)
char *string;

DESCRIPTION

System gives *string* to *sh*(1) as input, as if *string* had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

FILES

/bin/sh

SEE ALSO

exec(2), sh(1).

DIAGNOSTICS

System forks to create a child process that in turn executes (via *exec*) */bin/sh* in order to execute *string*. If the fork or exec fails, *system* returns −1 and sets *errno*.

SUPPORT STATUS

Supported.

NAME

tgetent, tsysent, tgetnum, tgetflag, tgetstr, tgeterm, tgoto, tputs — terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tsysent(bp,name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

tgeterm(ttyname, termid)
char *ttyname, *termid;

char *
tgoto(cm, destcol, destline)
char *cm;
int destcol;
int destline;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base *termcap*(5). These are low level routines; see *curses*(3) for a higher level package.

tgetent

Tgetent extracts the entry for terminal *name* into the buffer at *bp*. *Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*.

Tgetent returns −1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well.

Tgetent looks in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type name is the same as the environment string TERM, *tgetent* uses the TERMCAP string instead of reading the termcap file.

If the TERMCAP variable value does begin with a slash, the string is used as a path name rather than */etc/termcap*. This can speed up entry into programs that call *tgetent*, as well as helping debug new terminal descriptions or to make one for your terminal if you cannot write the file */etc/termcap.*

tsysent

Tsysent works like *tgetent* only the environment strings TERM and TERMCAP are not used. In this case **name** is used as the entry to search for in */etc/termcap.*

tgetnum

Tgetnum gets the numeric value of capability *id,* returning −1 if *id* is not given for the terminal.

tgetflag

Tgetflag returns 1 if the specified capability is present in the entry for the terminal, 0 if it is not.

tgetstr

Tgetstr gets the string value of capability *id,* placing it in the buffer at *area,* advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap*(5), except for cursor addressing and padding information.

tgeterm

Tgeterm searches the file /etc/**ttytype** for an entry with matching tty port name. If an entry is found *tgeterm* copies the corresponding termcap data base type to the buffer *termid.* If no port name is found that matches, *tgeterm* uses *unknown*.

The file /etc/**ttytype** is typically used by the system administrator to define a termcap data base name for each terminal port connected to the system.

tgoto

Tgoto returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline.* It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **\n**, **^D** or **^@** in the returned string. If a % sequence is given which is not understood, then *tgoto* returns OOPS.

tputs

Tputs decodes the leading padding information of the string *cp; affcnt* is the number of lines affected by the operation, or 1 if this is not applicable; *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as encoded by *stty (2).* The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a null (**^@**) is inappropriate.

NOTE

Programs which call *tgoto* should be sure to turn off the XTABS bit(s), since *tgoto* may now output a tab. Programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.

FILES

usr/lib/libtermcap.a	termcap library
/etc/termcap	data base

/etc/ttytype data base of terminal types by port

SEE ALSO

curses(3), termcap(5), ex(1).

SUPPORT STATUS

Supported.

NAME

tmpfile − create a temporary file

SYNOPSIS

#include <stdio.h>

FILE *tmpfile ()

DESCRIPTION

Tmpfile creates a temporary file opened for update and returns a corresponding FILE pointer. The file is automatically deleted when the process using it terminates.

SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C), tmpnam(3S).

SUPPORT STATUS

Supported.

NAME

tmpnam, tempnam — create a name for a temporary file

SYNOPSIS

#include <stdio.h>

char *tmpnam (s)
char *s;

char *tempnam (dir, pfx)
char *dir, *pfx;

DESCRIPTION

These functions generate file names that can safely be used for a temporary file.

Both functions generate a different file name each time they are called.

Files created using these functions and either *fopen* or *creat* are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the responsibility of the user to use *unlink* (2) to remove the file when its use is ended.

tmpnam

Tmpnam always generates a file name using the path-name defined as **P_tmpdir** in the *<stdio.h>* header file. If *s* is NULL, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to *tmpnam* destroys the contents of the area. If *s* is not NULL, *s* is assumed to be the address of an array of at least **L_tmpnam** bytes, where **L_tmpnam** is a constant defined in *<stdio.h>*; *tmpnam* places its result in that array and returns *s*.

tempnam

Tempnam allows the user to control the choice of a directory. The argument *dir* points to the path-name of the directory in which the file is to be created. If *dir* is NULL or points to a string which is not a path-name for an appropriate directory, the path-name defined as **P_tmpdir** in the *<stdio.h>* header file is used. If that path-name is not accessible, **/tmp** is used. This entire sequence can be overridden by providing an environment variable **TMPDIR** in the user environment; the value of **TMPDIR** is a path-name for the desired temporary-file directory.

Many applications prefer the names of their temporary files to have certain particular initial letter sequences. Use the *pfx* argument fo specify these. *Pfx* may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

Tempnam uses *malloc*(3C) to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from *tempnam* may serve as an argument to *free* (see *malloc*(3C)). If *tempnam* cannot return the expected result for any reason, i.e. *malloc* failed, or none of the above mentioned attempts to find an appropriate directory was successful, *tempnam* returns a NULL pointer.

SEE ALSO

creat(2), unlink(2), fopen(3S), malloc(3C), mktemp(3C), tmpfile(3S).

RESTRICTIONS

If called more than 17,576 times in a single process, these functions recycle previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This never happens if the other process uses these functions or *mktemp*, and the file names are chosen so as to render duplication by other means unlikely.

SUPPORT STATUS

Supported.

NAME

sin, dsin, csin, cos, dcos, ccos, tan, dtan, asin, dasin, acos, dacos, atan, datan, atan2, datan2 − Fortran trigonometric intrinsic functions

SYNOPSIS

```
real r1, r2, r3
double precision dp1, dp2, dp3
complex cx1, cx2

r2 = sin(r1)

dp2 = dsin(dp1)
dp2 = sin(dp1)

cx2 = csin(cx1)
cx2 = sin(cx1)

r2 = cos(r1)

dp2 = dcos(dp1)
dp2 = cos(dp1)

cx2 = ccos(cx1)
cx2 = cos(cx1)

r2 = tan(r1)

dp2 = dtan(dp1)
dp2 = tan(dp1)

r2 = asin(r1)

dp2 = dasin(dp1)
dp2 = asin(dp1)

r2 = acos(r1)

dp2 = dacos(dp1)
dp2 = acos(dp1)

r2 = atan(r1)

dp2 = datan(dp1)
dp2 = atan(dp1)

r3 = atan2(r1, r2)

dp3 = datan2(dp1, dp2)
dp3 = atan2(dp1, dp2)
```

DESCRIPTION

Sin, the generic sine function, returns the sine of its argument in radians; the returned value is the same type as the argument.

Dsin returns the double-precision sine of its double-precision argument.

Csin returns the complex sine of its complex argument.

Cos, the generic cosine function, returns the cosine of its argument in radians; the returned value is the same type as the argument.

Dcos returns the double-precision cosine of its double-precision argument.

Ccos returns the complex cosine of its complex argument.

Tan, the generic tangent function, returns the tangent of its argument in radians; the returned value is the same type as the argument.

Dtan returns the double-precision tangent of its double-precision argument.

Asin, the generic arcsine function, returns the arcsine of its argument in radians; the returned value is the same type as the argument.

Dasin returns the double-precision arcsine of its double-precision argument.

Acos, the generic arccosine function, returns the arccosine of its argument in radians; the returned value is the same type as the argument.

Dacos returns the double-precision arccosine of its double-precision argument.

Atan, the generic arctangent function, returns the arctangent of its argument in radians; the returned value is the same type as the argument.

Datan returns the double-precision arctangent of its double-precision argument.

Atan2, the generic form, returns the arctangent of *arg1/arg2*; the returned value is the same type as the arguments.

Datan2 returns the double-precision arctangent of its double-precision arguments.

SEE ALSO
trig(3M).

SUPPORT STATUS
Not supported.

NAME

sin, cos, tan, asin, acos, atan, atan2 — trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin (x)
double x;

double cos (x)
double x;

double tan (x)
double x;

double asin (x)
double x;

double acos (x)
double x;

double atan (x)
double x;

double atan2 (y, x)
double x, y;
```

DESCRIPTION

Sin, *cos* and *tan* return respectively the sine, cosine and tangent of their argument, which is in radians.

Asin returns the arcsine of x, in the range $-\pi/2$ to $\pi/2$.

Acos returns the arccosine of x, in the range 0 to π.

Atan returns the arctangent of x, in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arctangent of y/x, in the range $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUE

Sin, *cos* and *tan* lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return 0 when there would otherwise be a complete loss of significance. In this case a message indicating TLOSS error is printed on the standard error output. For less extreme arguments, a PLOSS error is generated but no message is printed. In both cases, *errno* is set to **ERANGE.**

Tan returns **HUGE** for an argument which is near an odd multiple of $\pi/2$ when the correct value would overflow, and sets *errno* to **ERANGE.**

When passed arguments of magnitude greater than 1.0, *asin* and *acos* return 0 and set *errno* to **EDOM.** In addition, a message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

SUPPORT STATUS

Supported.

NAME

sinh, dsinh, cosh, dcosh, tanh, dtanh — Fortran hyperbolic intrinsic functions

SYNOPSIS

real r1, r2
double precision dp1, dp2

r2 = sinh(r1)

dp2 = dsinh(dp1)
dp2 = sinh(dp1)

r2 = cosh(r1)

dp2 = dcosh(dp1)
dp2 = cosh(dp1)

r2 = tanh(r1)

dp2 = dtanh(dp1)
dp2 = tanh(dp1)

DESCRIPTION

Sinh, the generic hyperbolic sine function, returns the hyperbolic sine of its argument; the type of the returned value is the same as the argument.

Dsinh returns the double-precision hyperbolic sine of its double-precision argument.

Cosh, the generic hyperbolic cosine function, returns the hyperbolic cosine of its argument; the value returned is the same type as the argument.

Dcosh returns the double-precision hyperbolic cosine of its double-precision argument.

Tanh, the generic hyperbolic tangent function, returns the hyperbolic tangent of its argument; the returned value is the same type as the argument.

Dtanh returns the double-precision hyperbolic tangent of its double precision argument.

SEE ALSO

trigh(3M).

SUPPORT STATUS

Not supported.

NAME

sinh, cosh, tanh – hyperbolic functions

SYNOPSIS

#include <math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;

DESCRIPTION

Sinh, *cosh* and *tanh* return respectively the hyberbolic sine, cosine and tangent of their argument.

DIAGNOSTICS

Sinh and *cosh* return **HUGE** when the correct value would overflow, and set *errno* to **ERANGE.**

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

SUPPORT STATUS

Supported.

NAME

tsearch, tdelete, twalk — manage binary search trees

SYNOPSIS

```
#include <search.h>

char *tsearch ((char *) key, (char **) rootp, compar)
int (*compar)( );

char *tfind ((char *) key, (char **) rootp, compar)
int (*compar)( );

char *tdelete ((char *) key, (char **) rootp, compar)
int (*compar)( );

void twalk ((char *) root, action)
void (*action)( );
```

DESCRIPTION

tsearch

Tsearch is a binary tree search routine generalized from Knuth (6.2.2) Algorithm T. It returns a pointer into a tree indicating where a datum may be found. If the datum does not occur, *tsearch* adds it at an appropriate point in the tree.

Key points to the datum to be sought in the tree.

Rootp points to a variable that points to the root of the tree. A NULL pointer value for *rootp* denotes an empty tree. If *rootp* is NULL, tsearch sets *rootp* to point to the datum at the root of the new tree.

Compar is the name of a user supplied comparison function. It is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

tfind

Tfind searches for a datum in if found. If it is not found, *tfind* returns a NULL pointer. The arguments for *tfind* are the same as for *tsearch*.

tdelete

Tdelete deletes a node from a binary search tree. It is generalized from Knuth (6.2.2) algorithm D. The arguments are the same as for *tsearch*. The variable pointed to by *rootp* is changed if the deleted node was the root of the tree. *Tdelete* returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

twalk

Twalk traverses a binary search tree.

Root is the root of the tree to be traversed. Any node in a tree may be used as the root for a walk below that node.

Action is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type

typedef enum { preorder, postorder, endorder, leaf } VISIT;

defined in the <search.h> header file, depending on whether the node is a leaf, or whether this is the first, second or third time that the node has been visited during a depth-first, left-to-right traversal of the tree. The third argument is the level of the node in the tree; the root is level zero.

NOTES

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

RETURN VALUE

A NULL pointer is returned by *tsearch* if there is not enough space available to create a new node.

A NULL pointer is returned by *tsearch* and *tdelete* if *rootp* is NULL on entry.

If the datum is found, both *tsearch* and *tfind* return a pointer to it. If not *tfind* returns NULL, and *tsearch* returns a pointer to the inserted item.

EXAMPLE

Example 1

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

struct node {   /* pointers to these are stored in the tree */
        char *string;
        int length;
};
char string_space[10000];       /* space to store strings */
struct node nodes[500];                 /* nodes to store */
struct node *root = NULL;       /* this points to the root */

main( )
{
        char *strptr = string_space;
        struct node *nodeptr = nodes;
        void print_node( ), twalk( );
        int i = 0, node_compare( );
```

```
          while (gets(strptr) != NULL && i++ < 500)  {
                    /* set node */
                    nodeptr->string = strptr;
                    nodeptr->length = strlen(strptr);
                    /* put node into the tree */
                    (void) tsearch((char *)nodeptr, &root,
                              node_compare);
                    /* adjust pointers, so we don't overwrite tree */
                    strptr += nodeptr->length + 1;
                    nodeptr++;
          }
          twalk(root, print_node);
}
/*
```

Example 2

This routine compares two nodes, based on an alphabetical ordering of the string field.

```
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
          return strcmp(node1->string, node2->string);
}
/*
```

Example 3

This routine prints out a node, the first time *twalk* encounters it.

```
*/
void
print_node(node, order, level)
struct node **node;
VISIT order;
int level;
{
          if (order == preorder  order == leaf)  {
                    (void)printf("string = %20s,  length = %d\n",
                              (*node)->string, (*node)->length);
          }
}
```

SEE ALSO

bsearch(3C), hsearch(3C), lsearch(3C).

WARNING

The *root* argument to *twalk* is one level of indirection less than the *rootp* arguments to *tsearch* and *tdelete*.

If the calling function alters the pointer to the root, severe damage may occur.

SUPPORT STATUS

Supported.

NAME

ttyname, isatty — find name of a terminal

SYNOPSIS

char *ttyname (fildes)
int fildes;

int isatty (fildes)
int fildes;

DESCRIPTION

Ttyname returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor *fildes*.

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

FILES

/dev/*

DIAGNOSTICS

Ttyname returns a NULL pointer if *fildes* does not describe a terminal device in directory /dev.

RESTRICTIONS

The return value points to static data whose content is overwritten by each call.

SUPPORT STATUS

Supported.

NAME

ttyslot — find the slot in the utmp file of the current user

SYNOPSIS

int ttyslot ( )

DESCRIPTION

Ttyslot returns the index of the current user entry in the **/etc/utmp** file. *Ttyslot* actually scans the file **/etc/inittab** for the name of the terminal associated with the standard input, the standard output, or the error output (standard files 0, 1 or 2).

FILES

/etc/inittab
/etc/utmp

SEE ALSO

getut(3C), ttyname(3C).

RETURN VALUE

Ttyslot returns 0 if an error is encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

SUPPORT STATUS

Supported.

NAME

ungetc — push character back into input stream

SYNOPSIS

#include <stdio.h>

int ungetc (c, stream)
char c;
FILE *stream;

DESCRIPTION

Ungetc inserts the character *c* into the buffer associated with an input *stream*. That character, *c*, is returned by the next *getc* call on that *stream*. *Ungetc* returns *c*, and leaves the file *stream* unchanged.

In order that *ungetc* perform correctly, a read statement must have been performed prior to the call of the *ungetc* function.

Ungetc returns **EOF** if it cannot insert the character. In the case that *stream* is *stdin*, *ungetc* allows exactly one character to be pushed back onto the buffer without a previous read statement.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered.

If *c* equals **EOF**, *ungetc* does nothing to the buffer and returns **EOF**.

Fseek(3S) erases all memory of inserted characters.

DIAGNOSTICS

Ungetc returns **EOF** if it can not insert the character.

SEE ALSO

fseek(3S), getc(3S), setbuf(3S).

SUPPORT STATUS

Supported.

NAME

vprintf, vfprintf, vsprintf − print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf (format, ap)
char *format;
va_list ap;

int vfprintf (stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf (s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

vprintf, *vfprintf*, and *vsprintf* are the same as *printf*, *fprintf*, and *sprintf* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by *varargs*(5).

EXAMPLE

The following demonstrates how *vfprintf* could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
        .
        .
        .
/*
 * error should be called like
 *      error(function_name, format, arg1, arg2...);
 */
/*VARARGS0*/
void
error(va_alist)
/* Note that the function_name and format arguments cannot
 * be separately declared because of the definition of varargs.
 */
va_dcl
{
  va_list args;
  char *fmt;

  va_start(args);
  /* print out name of function causing error */
  (void)fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
  fmt = va_arg(args, char *);
  /* print out remainder of message */
  (void)vfprintf(fmt, args);
```

```
    va_end(args);
    (void)abort( );
}
```

SEE ALSO

vprintf(3X), varargs(5).

SUPPORT STATUS

Supported.

NAME

vprintf, vfprintf, vsprintf − print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf (format, ap)
char *format;
va_list ap;

int vfprintf (stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf (s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

vprintf, *vfprintf*, and *vsprintf* are the same as *printf*, *fprintf*, and *sprintf* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by *varargs*(5).

EXAMPLE

The following demonstrates how *vfprintf* could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
        .
        .
        .
/*
 *      error should be called like
 *      error(function_name, format, arg1, arg2...);
 */
/*VARARGS0*/
void
error(va_alist)
/* Note that the function_name and format arguments cannot
 * be separately declared because of the definition of varargs.
 */
va_dcl
{
  va_list args;
  char *fmt;

  va_start(args);
  /* print out name of function causing error */
  (void)fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
  fmt = va_arg(args, char *);
  /* print out remainder of message */
  (void)vfprintf(fmt, args);
```

```
    va_end(args);
    (void)abort( );
}
```

SEE ALSO

printf(3S), varargs(5).

SUPPORT STATUS

Supported.

NAME

intro – introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C **struct** declarations for the file formats are given where applicable. Usually, these structures can be found in the directories **/usr/include** or **/usr/include/sys.**

SUPPORT STATUS

Supported.

NAME

a.out — common assembler and link editor output

DESCRIPTION

The file name **a.out** is the output file from the assembler *as*(1) and the link editor *ld*(1). Both programs make *a.out* executable if there were no errors in assembling or linking and no unresolved external references.

ORGANIZATION

A common object file consists of a file header, an UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
UNIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last four sections (relocation, line numbers, symbol table, and string table) may be missing if the program was linked with the **−s** option of *ld*(1) or if the symbol table and relocation bits were removed by *strip*(1). Also note that the relocation information is absent after linking unless the **−r** option of *ld(1)* was used. The string table exists only if the symbol table contains symbols with names longer than eigth characters.

The sizes of each segment (contained in the header, discussed below) are in bytes and are even.

Unlinked object modules, as output from the assembler, do not have a UNIX system header.

LOAD ADDRESSES

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all zeros), and a stack.

TOWER 32 Systems

The text segment begins at location 0 in the core image. The header is not loaded.

If the magic number (the first field in the UNIX system header) is 0407 (octal), the text segment may not be write-protected or shared, so the data segment is contiguous with the text segment. If the magic number is 0410 (octal), the data segment and the text segment are not writable by the program; if other processes are executing the same **a.out** file, they share a single text segment.

The stack begins just below E00000 (hex) and grows toward lower addresses; the user program text and data begin at user address zero.
The stack is automatically extended as required.

TOWER XP Systems

The text segment begins at location 0x8000 in the core image. The header is not loaded.

If the magic number (the first field in the UNIX system header) is 0407 (octal), the text segment may not be write-protected or shared, so the data segment is contiguous with the text segment. If the magic number is 0410 (octal), the data segment and the text segment are not writable by the program; if other processes are executing the same **a.out** file, they share a single text segment.

The stack begins just below 108000 (hex) and grows toward lower addresses; the user program text and data begin at 8000 (hex). The stack is automatically extended as required.

The data segment is extended only as requested by the *brk*(2) system call.

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that appears in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, the storage class of the symbol-table entry for that word is marked as an external symbol, and the section number is set to 0. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to the word in the file.

CONTENTS

File Header

The format of the **filehdr** header is

```
struct filehdr
{
        unsigned short  f_magic;   /* magic number */
        unsigned short  f_nscns;   /* number of sections */
        long            f_timdat;  /* time and date stamp */
        long            f_symptr;  /* file ptr to symtab */
        long            f_nsyms;   /* # symtab entries */
        unsigned short  f_opthdr;  /* sizeof(opt hdr) */
        unsigned short  f_flags;   /* flags */
};
```

UNIX Header

The format of the UNIX system header is

```
typedef struct aouthdr {
        short   magic;          /* magic number */
        short   vstamp;         /* version stamp */
        long    tsize;          /* text size in bytes, padded */
        long    dsize;          /* initialized data (.data) */
        long    bsize;          /* uninitialized data (.bss) */
        long    entry;          /* entry point */
        long    text_start;     /* base of text used for this file */
        long    data_start;     /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;   /* physical address */
        long            s_vaddr;   /* virtual address */
        long            s_size;    /* section size */
        long            s_scnptr;  /* file ptr to raw data */
        long            s_relptr;  /* file ptr to relocation */
        long            s_lnnoptr; /* file ptr to line numbers */
        unsigned short  s_nreloc;  /* # reloc entries */
        unsigned short  s_nlnno;   /* # line number entries */
        long            s_flags;   /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it is in the following format:

```
struct reloc
{
        long    r_vaddr;   /* (virtual) address of reference */
        long    r_symndx;  /* index into symbol table */
        short   r_type;    /* relocation type */
};
```

The start of the relocation information is *relptr* from the Section Header.
If there is no relocation information, *relptr* is 0.

Symbol Table

The format of the symbol table header is

```
#define  SYMNMLEN  8
#define  FILNMLEN  14
#define  SYMESZ    18          /* the size of a SYMENT */

struct syment
{
   union                             /* get a symbol name */
   {
      char          _n_name[SYMNMLEN]; /* name of symbol */
      struct
      {
         long       _n_zeroes;     /* == 0L if in string table */
         long       _n_offset;     /* location in string table */
      } _n_n;
      char          *_n_nptr[2];   /* allows overlaying */
   } _n;
   unsigned long    n_value;       /* value of symbol */
   short            n_scnum;       /* section number */
   unsigned short   n_type;        /* type and derived type */
   char             n_sclass;      /* storage class */
   char             n_numaux;      /* number of aux entries */
};

#define  n_name      _n._n_name
#define  n_zeroes    _n._n_n._n_zeroes
#define  n_offset    _n._n_n._n_offset
#define  n_nptr      _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
        struct {
                long    x_tagndx;
                union {
                        struct {
                         unsigned short         x_lnno;
                         unsigned short         x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union {
                        struct {
                         long    x_lnnoptr;
                         long    x_endndx;
                        }  x_fcn;
                        struct {
                         unsigned short   x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
```

```
                unsigned short  x_tvndx;
        } x_sym;

        struct {
                char    x_fname[FILNMLEN];
        } x_file;

        struct {
                long            x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

SEE ALSO

brk(2), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4), as(1), cc(1), ld(1).

SUPPORT STATUS

Supported.

NAME

acct − per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced by calling *acct*(2) have records in the form defined by <sys/acct.h>, whose contents are:

```
typedefushort comp_t;  /* "floating point" */
                       /* 13-bit fraction, 3-bit exponent  */

struct  acct
{
        char    ac_flag;        /* Accounting flag */
        char    ac_stat;        /* Exit status */
        ushort  ac_uid;
        ushort  ac_gid;
        dev_t   ac_tty;
        time_t  ac_btime;       /* Beginning time */
        comp_t  ac_utime;       /* acctng user time in clock ticks */
        comp_t  ac_stime;       /* acctng system time in clock
                                ticks */
        comp_t  ac_etime;       /* acctng elapsed time in clock
                                ticks */
        comp_t  ac_mem;         /* memory usage in clicks */
        comp_t  ac_io;          /* chars trnsfrd by read/write */
        comp_t  ac_rw;          /* number of block reads/writes */
        char    ac_comm[8];     /* command name */
};

extern  struct  acct            acctbuf;
extern  struct  inode           *acctp;  /* inode of accounting file */

#define AFORK   01              /* has executed fork, but no exec */
#define ASU     02              /* used super-user privileges */
#define ACCTF   0300            /* record type: 00 = acct */
```

In *ac_flag*, the AFORK flag is turned on by each *fork*(2) and turned off by an *exec*(2).

The *ac_comm* field is inherited from the parent process and is reset by any *exec*.

Each time the system charges the process with a clock tick, it also adds to *ac_mem* the current process size, computed as follows:

> (data size) + (text size) / (number of in-core processes using text)

The value of *ac_mem/(ac_stime + ac_utime)* can be viewed as an approximation to the mean process size, as modified by text-sharing.

The structure **tacct.h**, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/* total accounting (for acct period), also for day */

struct tacct {
        uid_t           ta_uid;       /* userid */
        char            ta_name[8];   /* login name */
        float           ta_cpu[2];    /* cum. cpu time, p/np (mins) */
        float           ta_kcore[2];  /* cum kcore-minutes, p/np */
        float           ta_con[2];    /* cum. connect time, p/np,
                                         mins */
        float           ta_du;        /* cum. disk usage */
        long            ta_pc;        /* count of processes */
        unsigned short  ta_sc;        /* count of login sessions */
        unsigned short  ta_dc;        /* count of disk samples */
        unsigned short  ta_fee;       /* fee for special services */
};
```

SEE ALSO

acct(2), exec(2), fork(2), acct(1M), acctcom(1).

RESTRICTIONS

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

SUPPORT STATUS

Not supported.

NAME

alert{mmddyy} – error records for devices exceeding threshold values

DESCRIPTION

The */ncrm/checklog/alert{mmddyy}* file, where {mmddyy} is the month, day, and year on which the file was created, is created by *logalert*(1M). The file contains the error records from the error log for devices which exceeded set threshold values. This file is in the same format as the error log file (see *errfile*(4).

To examine these records execute the command

```
errpt –a –f /ncrm/checklog/alert{mmddyy} | more
```

specifying the appropriate *alert{mmddyy}* file.

Error records for some devices which have exceeded their threshold values may not exist in the */ncrm/checklog/alert{mmddyy}* file because the transfer flag was turned off in the */ncrm/checklog/threshold* file (see *threshold*(4). To examine these error log records in the error log file that was used when executing *logalert*, execute the command

```
errpt –d <device name> –f <file> | more
```

specifing the device name and the error file that was used when executing *logalert*, if any.

SEE ALSO

logalert(1M), errpt(1M), errfile(4), threshold(4), alertmesg(4).

SUPPORT STATUS

Supported.

NAME

alertmesg — logalert summary message file

DESCRIPTION

The */ncrm/checklog/alertmesg* file contains the messages generated by *logalert* when it detects devices that have errors exceeding set threshold values. This is an ASCII file.

Each entry contains the following information:

Date device exceeded threshold value
Name of the error log file from which the errors were tallied
The name of the device that exceeded its threshold
The number of errors for this device that were found in the error file
The threshold value for the device

In the case of a file device with bad blocks, the following information is also given:

The logical device name
The block threshold
The number of bad blocks detected on the device
The specific bad block numbers

For example:

```
DATE=09/04/85  11:00:04  file=/usr/adm/errfile
  Device=H501  Device Errors=15  Error Threshold=10

DATE=12/04/85  15:36:50  file=errfile3.ecc
  Device=H501  Device Errors=58  Error Threshold=15
  Block Threshold=11  Bad Blocks:
  blk=56728  errors=20;  blk=56789  errors=19;
  blk=56777  errors=19;
```

SEE ALSO

logalert(1M), threshold(4), alert(4).

SUPPORT STATUS

Supported.

NAME

ar − common archive file format

DESCRIPTION

The archive command *ar*(1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld*(1).

Each archive begins with the archive magic string.

```
#define ARMAG  "!<arch>\n"        /* magic string */
#define SARMAG 8                  /* length of magic string */
```

Each archive which contains common object files (see *a.out*(4)) includes an archive symbol table. This symbol table is used by the link editor *ld*(1) to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by *ar*.

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define ARFMAG  "`\n"         /* header trailer string */

struct ar_hdr                 /* file member header */
{
 char ar_name[16];            /* '/' terminated file member name */
 char ar_date[12];            /* file member date */
 char ar_uid[6];              /* file member user identification */
 char ar_gid[6];              /* file member group identification */
 char ar_mode[8];             /* file member mode (octal) */
 char ar_size[10];            /* file member size */
 char ar_fmag[2];             /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The *ar_name* field is blank-padded and slash (/) terminated. The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command *ar*(1) is used. Conversion tools such as *arcv*(1) and *convert*(1) exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0] ==** '/'). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".
- The name string table. Length: *ar_size* − (4 bytes * ("the number of symbols" + 1)).

The number of symbols and the array of offsets are managed with *sgetl* and *sputl*. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO

sputl(3X), a.out(4), ar(1), arcv(1), convert(1), ld(1), strip(1).

WARNINGS

Strip(1) removes all archive symbol entries from the header. The archive symbol entries must be restored via the ts option of the *ar*(1) command before the archive can be used with the link editor *ld*(1).

SUPPORT STATUS

Supported.

NAME

checklist – list of file systems processed by fsck

DESCRIPTION

Checklist resides in directory /etc and contains a list of at most 15 *special file* names. Each *special file* name is on a separate line and corresponds to a file system. Each file system is then automatically processed by the *fsck*(1M) command.

SEE ALSO

fsck(1M).

SUPPORT STATUS

Supported.

NAME

core — format of core image file

DESCRIPTION

UNIX writes out a core image of a terminated process when any of various errors occur. See *signal*(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the working directory of the process, if possible; normal access controls apply. A process with an effective user ID different from the real user ID does not produce a core image.

The first section of the core image is a copy of the system per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in **/usr/include/sys/param.h**.

The remainder represents the actual contents of the core area of the user when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **/usr/include/sys/user.h**. The locations of the registers are outlined in **/usr/include/sys/reg.h**.

SEE ALSO

setuid(2), signal(2), crash(1M), sdb(1).

SUPPORT STATUS

Supported.

NAME

cpio — format of cpio archive

DESCRIPTION

The *header* structure, when the —c option of *cpio*(1) is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        ushort  h_ino,
                h_mode,
                h_uid,
                h_gid;
        short   h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to word];
} Hdr;
```

When the —c option is used, the *header* information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
      &Hdr.h_magic,
      &Hdr.h_dev,
      &Hdr.h_ino,
      &Hdr.h_mode,
      &Hdr.h_uid,
      &Hdr.h_gid,
      &Hdr.h_nlink,
      &Hdr.h_rdev,
      &Longtime,
      &Hdr.h_namesize,
      &Longfile,
      Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The content of each file is recorded in an element of the array of varying length structures, archived together with other items describing the file.

Every instance of *h_magic* contains the constant 070707 (octal).

The items *h_dev* through *h_mtime* have meanings explained in *stat*(2).

The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name **TRAILER!!!**. Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

SEE ALSO

stat(2), cpio(1), find(1).

SUPPORT STATUS

Supported.

NAME

dir — format of directories

SYNOPSIS

#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, but no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry (see *fs*(4)).

The structure of a directory entry as given in the include file is:

```
#ifndef DIRSIZ
#define DIRSIZ     14
#endif
struct  direct
{
        ino_t d_ino;
        char  d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for . and ... The first . is an entry for the directory itself. The second .. is an entry for the parent directory.

The meaning of .. is modified for the root directory of the master file system; there is no parent, so .. has the same meaning as ..

SEE ALSO

fs(4).

SUPPORT STATUS

Supported.

NAME

errfile — error-log file format

DESCRIPTION

When the system detects hardware errors, it generates an error record and passes that record to the error-logging daemon, which records the error in the error log for later analysis. The default error log is **/usr/adm/errfile**.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```
struct errhdr {
        short           e_type;         /* record type */
        short           e_len;          /* bytes in record (with header) */
        time_t          e_time;         /* time of day */
};
```

The permissible record types are as follows:

```
#define E_GOTS    010     /* start for the UNIX/TS */
#define E_GORT    011     /* start for the UNIX/RT */
#define E_STOP    012     /* stop */
#define E_TCHG    013     /* time change */
#define E_CCHG    014     /* configuration change */
#define E_BLK     020     /* block device error */
#define E_STRAY   030     /* stray interrupt */
#define E_PRTY    031     /* memory parity */
```

Some records in the error file are of an administrative nature. These include the startup record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated gracefully, and the time-change record that is used to account for changes in the system time-of-day. These records have the following formats:

```
struct estart {
        short           e_cpu;          /* CPU type */
        struct utsname  e_name;         /* system names */
        short           e_mmr3;         /* contents mem mgmt reg 3  */
        long            e_syssize;      /* system memory size  */
        short           e_bconf;        /* block dev configuration */
};

#define eend errhdr  /* record header */

struct etimchg {
        time_t          e_ntime;        /* new time */
};
```

Stray interrupts cause a record with the following format to be logged:

```
struct estray {
      physadr        e_saddr;     /* stray loc or device addr */
      short          e_sbacty;    /* active block devices */
};
```

Error records for block devices have the following format:

```
struct eblock {

      dev_t          e_dev;       /* true major + minor dev no */
      physadr        e_regloc;    /* controller address */
      short          e_bacty;     /* other block I/O activity */
      struct iostat {
        long         io_ops;      /* number read/writes */
        long         io_misc;     /* number other operations */
        ushort       io_unlog;    /* number unlogged errors */
      }              e_stats;
      short          e_bflags;    /* read/write, error, etc */
      short          e_cyloff;    /* logical dev start cyl */
      daddr_t        e_bnum;      /* logical block number */
      ushort         e_bytes;     /* number bytes to transfer */
      paddr_t        e_memadd;    /* buffer memory address */
      ushort         e_rtry;      /* number retries */
      short          e_nreg;      /* number device registers */
#ifdef vax
      struct mba_regs {
        long mba_csr;
        long mba_cr;
        long mba_sr;
        long mba_var;
        long mba_vcr;
      } e_mba;
#endif
};
```

The following values are used in the *e_bflags* word:

```
#define E_WRITE    0      /* write operation */
#define E_READ     1      /* read operation */
#define E_NOIO     02     /* no I/O pending */
#define E_PHYS     04     /* physical I/O */
#define E_MAP      010    /* Unibus map in use */
#define E_ERROR    020    /* I/O failed */
```

The *true* major device numbers that identify the failing device are as follows:

Digital Equipment		*Western Electric*	
#define RK0	0	#define DFC0	0
#define RP0	1	#define IOP0	1
#define RF0	2	#define MT0	2
#define TM0	3		
#define TC0	4		
#define HP0	5		
#define HT0	6		
#define HS0	7		
#define RL0	8		
#define HP1	9		
#define HP2	10		
#define HP3	11		

SEE ALSO

errdemon(1M).

SUPPORT STATUS

Supported.

NAME

filehdr − file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct  filehdr
{
        unsigned short  f_magic ;   /* magic number */
        unsigned short  f_nscns ;   /* number of sections */
        long            f_timdat ;  /* time & date stamp */
        long            f_symptr ;  /* file ptr to symtab */
        long            f_nsyms ;   /* # symtab entries */
        unsigned short  f_opthdr ;  /* sizeof(opt hdr) */
        unsigned short  f_flags ;   /* flags */
} ;
```

F_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(3S) to position an I/O stream to the symbol table. The UNIX System optional header on the system is always 28 bytes in length. The valid magic numbers are given below:

```
#define NCRWRMAGIC 0610    /* Tower writable text
                              segments */
#define NCRROMAGIC  0615   /* Tower readonly sharable
                              text segments */
```

The value in *f_timdat* is obtained from the *time*(2) system call. Flag bits currently defined are:

```
#define F_RELFLG  00001   /* relocation entries stripped */
#define F_EXEC    00002   /* file is executable */
#define F_LNNO    00004   /* line numbers stripped */
#define F_LSYMS   00010   /* local symbols stripped */
#define F_MINMAL  00020   /* minimal object file */
#define F_UPDATE  00040   /* update file, ogen produced */
#define F_SWABD   00100   /* file is "pre-swabbed" */
#define F_AR16WR  00200   /* 16 bit DEC host */
#define F_AR32WR  00400   /* 32 bit DEC host */
#define F_AR32W   01000   /* non-DEC host */
#define F_PATCH   02000   /* "patch" list in opt hdr */
```

SEE ALSO

time(2), fseek(3S), a.out(4).

SUPPORT STATUS

Supported.

NAME

file system − format of system volume

SYNOPSIS

#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>

DESCRIPTION

Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512 byte long sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the *super-block*. The format of a super-block is:

```
/*
 * Structure of the super-block
 */
struct  filsys
{
        ushort   s_isize;            /* size in blocks of i-list */
        daddr_t  s_fsize;            /* size in blocks of entire
                                        volume */
        short    s_nfree;            /* number of addresses
                                        in s_free */
        daddr_t  s_free[NICFREE];    /* free block list */
        short    s_ninode;           /* number of i-nodes in
                                        s_inode */
        ino_t    s_inode[NICINOD];   /* free i-node list */
        char     s_flock;            /* lock during free list
                                        manipulation */
        char     s_ilock;            /* lock during i-list
                                        manipulation */
        char     s_fmod;             /* super block modified
                                        flag */
        char     s_ronly;            /* mounted read-only flag */
        time_t   s_time;             /* last super block update */
        short    s_dinfo[4];         /* device information */
        daddr_t  s_tfree;            /* total free blocks*/
        ino_t    s_tinode;           /* total free inodes */
        char     s_fname[6];         /* file system name */
        char     s_fpack[6];         /* file system pack name */
        long     s_fill[13];         /* ADJUST to make size of
                                        filsys be 512 */
        long     s_magic;            /* magic number to indicate
                                        new file system */
        long     s_type;             /* type of new file system */
};

#define FsMAGIC 0xfd187e20           /* s_magic number */

#define Fs1b     1                   /* 512 byte block */
#define Fs2b     2                   /* 1024 byte block */
```

s_type and s_magic

S_type indicates the file system type. Currently, two types of file systems are supported: the original 512-byte oriented and the new improved 1024-byte oriented. *S_magic* is used to distinguish the original 512-byte oriented file systems from the newer file systems. If this field is not equal to the magic number, *FsMAGIC*, the type is assumed to be *Fs1b*, otherwise the *s_type* field is used.

A block is determined by the type. For the original 512-byte oriented file system, a block is 512 bytes. For the 1024-byte oriented file system, a block is 1024 bytes or two sectors. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

s_isize and s_fsize

S_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is *s_isize*−2 blocks long. *S_fsize* is the first block not potentially available for allocation to a file. The system uses these two numbers to check for bad block numbers; if an impossible block number is allocated from the free list or is freed, the system writes a diagnostic on the on-line console. Moreover, the free array is cleared, in order to prevent further allocation from a presumably corrupted free list.

s_free, s_nfree, and s_tfree

S_free, *s_nfree*, and *s_tfree* are used to maintain *free list* for each volume. *S_tfree* is the total free blocks available in the file system. The *s_free* array contains up to 49 numbers denoting free blocks in *s_free*[1], ..., *s_free*[*s_nfree*−1]. *S_free*[0] is the block number of the head of a chain of blocks constituting the *free*list; this chain is called the *free-chain*.

The first long integer in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 long integers of this chain member. The first of these 50 blocks is the link to the next member of the chain.

To allocate a block:

1. Decrement *s_nfree*; the new block is *s_free [s_nfree]*.
2. If the new block number is 0, there are no blocks left, so give an error.
3. If *s_nfree* is now 0,
   - read in the block named by the new block number
   - replace *s_nfree* by the first word of the block
   - copy the block numbers in the next 50 long integers into the *s_free* array

To free a block:

1. If *s_nfree* is 50,
   - copy *s_nfree* and the *s_free* array into the block

—

write the block out

—

set *s_nfree* to 0

2. Set *s_free*[*s_nfree*] to the number of the freed block.
3. Increment *s_nfree*.

s_inode, s_ninode, and s_tinode

S_ninode is the number of free i-numbers in the *s_inode* array.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an inode and its flags, see *inode*(4).

S_tinode is the total free inodes available in the file system.

To allocate an i-node:

1. If *s_ninode* is greater than 0,

   —

   decrement *s_ninode*

   —

   return *s_inode*[*s_ninode*].
2. If *s_inode* is 0,

   —

   read the i-list

   —

   place the numbers of all free inodes (up to 100) into the *s_inode* array

   —

   repeat from 1

To free an i-node:

1. If *s_ninode* is less than 100

   —

   place *s_ninode* into *s_inode*[*s_ninode*]

   —

   increment *s_ninode*
2. If *s_ninode* is 100

   —

   do not bother to enter the freed i-node into any table.

This list of i-nodes only speeds up the allocation process; whether the inode is actually free or not is information maintained in the inode itself.

s_flock and s_ilock

S_flock and *s_ilock* are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is likewise immaterial; *s_fmod* is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

s_ronly

S_ronly is a read-only flag to indicate write-protection.

s_time
S_time is the last time the super-block of the file system was changed, and is the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the *s_time* of the super-block for the root file system is used to set the system's idea of the time.

s_fname
S_fname is the name of the file system and *s_fpack* is the name of the pack.

FILES

/usr/include/sys/filsys.h
/usr/include/sys/stat.h

SEE ALSO

inode(4), fsck(1M), fsdb(1M), mkfs(1M).

SUPPORT STATUS

Supported.

NAME

fspec — format specification in text files

DESCRIPTION

A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file so that text files can be maintained on the UNIX system with non-standard tabs (tabs which are not set at every eighth column). Such files must usually be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets <: and :>. Each parameter consists of a keyletter, possibly followed immediately by a value.

PARAMETERS

Default values, which are assumed for parameters not supplied, are **t−8** and **m0**. If the first line of a file does not contain a format specification, these defaults are assumed for the entire file.

If the s parameter is not specified, no size checking is performed.

t*tabs* Specifies the tab settings for the file. *Tabs* must be one of the following: a list of column numbers separated by commas, indicating tabs set at the specified columns; a − followed immediately by an integer *n*, indicating tabs at intervals of *n* columns; or a − followed by the name of a standard tab specification.

Standard tabs are specified by **t−8**, or equivalently, **t1,9,17,25,**etc. The *tabs*(1) command defines which standard tabs are recognized.

s*size* Specifies a maximum line size. *Size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* Specifies a number of spaces to be prepended to each line. *Margin* must be an integer.

d Indicates that the line containing the format specification is to be deleted from the converted file.

e Indicates that the current format is to prevail only until another format specification is encountered in the file.

EXAMPLE

The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1), newform(1), send(1C), tabs(1).

SUPPORT STATUS

Supported.

NAME

gettydefs – speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) (see the Superuser Reference Manual) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by entering a <*break*> character.

Each entry in **/etc/gettydefs** has the following format:

label# initial-settings # final-settings # login-prompt\
#next-label

Each entry is followed by a blank line. Lines that begin with # are ignored and may be used to comment the file.

If *getty* is called without a file argument, then *getty* uses the first entry of **/etc/gettydefs**; thus the first entry of **/etc/gettydefs** is the default entry. The first entry is also used if *getty* cannot find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which brings up a terminal at 300 baud.

FIELDS

The fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as *nnn*, where *nnn* is the octal value of the desired character. The fields are:

label The string against which *getty* tries to match its second argument. It is often the speed, such as 1200, at which the terminal is supposed to run, but it does not need to be (see below).

initial-settings
The initial *ioctl*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. *Getty* understands the symbolic names specified in **/usr/include/sys/termio.h** (see *termio*(7) in the Superuser Reference Manual).

Normally only the speed flag is required in the *initial-settings*. *Getty* automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-settings* remain in effect until *getty* executes *login*(1).

final-settings
The final *ioctl*(2) settings, set just prior to *getty* executes *login*. These can assume the same values as the *initial-settings*. The speed flag is required. The composite setting SANE takes care of most of the other flags that need to be set so that the processor and terminal are communicating properly.

The other two commonly specified *final-settings* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL** which sends the sinal SIGHUP to the process to close (i.e.

hang up) the line.

login-prompt

The *login-prompt* to be printed. Unlike the above fields where white space is ignored (a space, tab or new-line), white space is included in the *login-prompt* field.

next-label

The next *label* of the entry in the table that *getty* should use if the user types a <*break*> or the input cannot be read. Usually, a series of speeds are linked together in this fashion, into a closed set. For instance, 2400 linked to 1200, which in turn is linked to 300, which finally is linked to 2400.

NOTE

It is strongly recommended that after making or modifying /etc/gettydefs, it be run through *getty* with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

ioctl(2), getty(1M), termio(7), login(1).

SUPPORT STATUS

Supported.

NAME

gps – graphical primitive string, format of graphical files

DESCRIPTION

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as *plot* (in *stat*(1G)) and *vtoc* (in *toc*(1G)) produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

lines The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point.

Parameters are available to set *color*, *weight*, and *style* (see below).

arc The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. Two points produce a line connecting the points; three points produce a circular arc through the points; more than three produce lines connecting the points.

Parameters are available to set *color, weight,* and *style*.

text The *text* primitive draws characters. The single required point locates the center of the first character to be drawn.

Parameters are *color*, *font*, *textsize*, and *textangle*.

hardware
The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

comment
A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS PARAMETERS

color An integer value set for *arc, lines,* and *text* primitives.

weight
An integer value set for *arc* and *lines* primitives to indicate line thickness:

0	narrow weight
1	bold
2	medium weight

style An integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on Tektronix 4010 series storage tubes:

0	solid
1	dotted
2	dot dashed
3	dashed
4	long dashed

font An integer value set for *text* primitives to designate the text font to be used in drawing a character string. *Font* is a four-bit *weight* value followed by a four-bit *style* value.

textsize
An integer value used in *text* primitives to express the size of the characters to be drawn. *Textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth this value in the size-orientation (*so*) word (see below).

textangle
A signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *Textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of its absolute value.

ORGANIZATION

GPS primitives are organized internally as follows:

lines	*cw points sw*
arc	*cw points sw*
text	*cw point sw so* [*string*]
hardware	*cw point* [*string*]
comment	*cw* [*string*]

cw *Cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s)
Point(s) is one or more pairs of integer coordinates. *Text* and *hardware* primitives only require a single *point*. *Point(s)* are values within a Cartesian plane or *universe* having 64K (−32K to +32K) points on each axis.

sw *Sw* is the style-word and is used in *lines, arc,* and *text* primitives. The first eight bits contain *color* information. In *arc* and *lines* the last eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive the last eight bits of *sw* contain the *font*.

so *So* is the size-orientation word used in *text* primitives. The first eight bits contain text size and the remaining eight bits contain text rotation.

string
String is a null-terminated character string. If the string does not end on a word boundary an additional null is added to the GPS file to insure word-boundary alignment.

SEE ALSO

graphics(1G), stat(1G), toc(1G).

SUPPORT STATUS

Not supported.

NAME

group − group file

DESCRIPTION

Group is an ASCII file which contains the following information for each group:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

The fields are separated by colons; each group is separated from the next by a new-line.

If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

FILES

/etc/group

SEE ALSO

crypt(3C), passwd(4), newgrp(1), passwd(1).

SUPPORT STATUS

Supported.

NAME

inittab – script for the init process

DESCRIPTION

The *inittab* file supplies the general process dispatching instructions to *init*. The process that constitutes the majority of the process dispatching activities of *init* is the line process */etc/getty* that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The *inittab* file is composed of entries that are position dependent and have the following format:

id:rstate:action:process

Each entry is delimited by a newline; however, a backslash preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted.

Comments may be inserted in the *process* field using the *sh*(1) convention for comments. Comments for lines that spawn *getty*s are displayed by the *who*(1) command. These comments should contain some information about the line such as the location.

There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file.

ENTRY FIELDS

id One to four characters which uniquely identify an entry.

rstate

The *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system.

Each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from **0** through **6**. For example, if the system is in *run-level* **1**, only those entries having a **1** in the *rstate* field are processed.

When *init* is requested to change *run-levels,* all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (SIGTERM) and allowed a 20 second grace period before being forcibly terminated by a kill signal (SIGKILL).

The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0**–**6**. If no *run-level* is specified, then *action* is taken on this *process* for all *run-levels* **0**–**6**.

There are three other values, **a**, **b** and **c**, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* (see *init*(1M)) process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that the system is only in these states for as long as it takes to execute all the entries associated with the states.

A process started by an **a**, **b** or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *SINGLE USER* state.

action

Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:

respawn

If the process does not exist then start the process. Do not wait for its termination, but continue scanning the *inittab* file. When the process dies restart it.

If the process currently exists then do nothing and continue scanning the *inittab* file.

wait

When *init* enters the *run-level* that matches the *rstate* of the entry, start the process and wait for its termination. All subsequent reads of the *inittab* file while *init* is in the same *run-level* cause *init* to ignore this entry.

once

When *init* enters a *run-level* that matches the *rstate* of the entry, start the process. Do not wait for its termination, when the process dies, do not restart it. If *init* enters a new *run-level* where the process is still running from a previous *run-level* change, the process is not restarted.

boot

Process this entry only at the boot-time read by *init* of the *inittab* file. *Init* starts the process, and does not wait for its termination. When the process dies, do not restart it.

In order for this instruction to be meaningful, the *rstate* should be the default or it must match the *run-level* of *init* at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait

Process the entry only at the boot-time read by *init* of the *inittab* file. Start the process, wait for its termination and, when the process dies, not restart it.

powerfail

Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR** see *signal*(2)).

powerwait

Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**). Wait until the process terminates before continuing any processing of *inittab*.

off

If the process associated with this entry is currently running, send the warning signal (SIGTERM) and wait 20 seconds before forcibly terminating the process with the kill signal (SIGKILL).

If the process is nonexistent, ignore the entry.

ondemand

This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. *Ondemand* is used only with the **a**, **b** or **c** values described in the *rstate* field.

initdefault

Scan the entry only when *init* is initially invoked.

Init uses this entry, if it exists, to determine which *run-level* to enter initially: *init* takes the highest *run-level* specified in the **rstate** field and uses that as its initial state. If the *rstate* field is empty, it is interpreted as **0123456** and so *init* enters *run-level* **6**.

Also, the **initdefault** entry can use **s** to specify that *init* starts in the *SINGLE USER* state. Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab**, then *init* requests an initial *run-level* from the user at reboot time.

sysinit

Execute this entry before *init* tries to access the console. This entry is only used to initialize devices on which *init* may try to ask the *run-level* question. These entries are executed and waited for before continuing.

process

Process is a *sh* command to be executed. The entire process field is prefixed with *exec* and passed to a forked *sh* as **sh −c** 'exec *command'*. For this reason, any legal *sh* syntax can appear in the the *process* field. Comments can be inserted with the ; *#comment* syntax.

FILES

/etc/inittab

SEE ALSO

exec(2), open(2), signal(2), getty(1M), init(1M), sh(1), who(1).

SUPPORT STATUS

Supported.

NAME

inode − format of an inode

SYNOPSIS

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

DESCRIPTION

An i-node for a plain file or directory in a file system has the following structure defined by <sys/ino.h>.

```
/* Inode structure as it appears on a disk block. */
struct dinode
{
        ushort  di_mode;        /* mode and type of file */
        short   di_nlink;       /* number of links to file */
        ushort  di_uid;         /* user id of owner */
        ushort  di_gid;         /* group id of owner */
        off_t   di_size;        /* number of bytes in file */
        char    di_addr[40];    /* disk block addresses */
        time_t  di_atime;       /* time last accessed */
        time_t  di_mtime;       /* time last modified */
        time_t  di_ctime;       /* time created */
};
/*
 * the 40 address bytes:
 *     39 used; 13 addresses
 *     of 3 bytes each.
 */
```

For the meaning of the defined types *off_t* and *time_t* see *types*(5).

FILES

/usr/include/sys/ino.h

SEE ALSO

stat(2), fs(4), types(5).

SUPPORT STATUS

Supported.

NAME

issue — issue identification file

DESCRIPTION

The file /etc/issue contains the *issue* or project identification to be printed as a login prompt. This ASCII file is read by *getty* and then written to any terminal spawned or respawned from the *lines* file.

FILES

/etc/issue

SEE ALSO

login(1).

SUPPORT STATUS

Supported.

NAME

ldfcn — common object file access routines

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

DESCRIPTION

The common object file access routines are a collection of functions for reading an object file that is in common object file form. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

STRUCTURE

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function *ldopen*(3X) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE *ldptr;

TYPE(ldptr) The file magic number, used to distinguish between archive members and simple object files.

IOPTR(ldptr) The file pointer returned by *fopen* and used by the standard input/output functions.

OFFSET(ldptr) The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.

HEADER(ldptr)
The file header structure of the object file.

FUNCTIONS

The object file access functions themselves may be divided into four categories:

functions that open or close an object file:

ldopen(3X)
ldaopen
open a common object file

ldclose(3X)
ldaclose
close a common object file

functions that read header or symbol table information:

ldahread(3X)
read the archive header of a member of an archive file

ldfhread(3X)
read the file header of a common object file

ldshread(3X)
ldnshread
read a section header of a common object file

ldtbread(3X)
read a symbol table entry of a common object file

ldgetname(3X)
retrieve a symbol name from a symbol table entry or from the string table

the function *ldtbindex*(3X) which returns the index of a particular common object file symbol table entry

functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section:

ldohseek(3X)
seek to the optional file header of a common object file

ldsseek(3X)
ldnsseek
seek to a section of a common object file

ldrseek(3X)
ldnrseek
seek to the relocation information for a section of a common object file

ldlseek(3X)
ldnlseek
seek to the line number information for a section of a common object file

ldtbseek(3X)
seek to the symbol table of a common object file

These functions are described in detail in their respective manual pages.

All the functions except *ldopen*, *ldaopen*, *ldgetname*, and *ldtbindex* return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h**. *Ldopen* and *ldaopen* both return pointers to a **LDFILE** structure.

MACROS

Additional access to an object file is provided through a set of macros defined in **ldfcn.h**. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the **LDFILE** structure into a reference to its file descriptor field.

The following macros are provided:

```
LDFILE      *ldptr;

GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)
```

The STROFFSET macro calculates the address of the string table in a system Release 2.01 object file. See the manual entries for the corresponding standard input/output library functions for details on the use of these macros.

The program must be loaded with the object file access routine library **libld.a**.

WARNING

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function *fseek*(3S). **FSEEK** should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members.

SEE ALSO

fseek(3S), ldahread(3X), ldclose(3X), ldgetname(3X), ldfhread(3X), ldlread(3X), ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldlseek(3X), ldshread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X), intro(5).
Common Object File Format in the Support Tools Guide.

SUPPORT STATUS

Supported.

NAME

linenum − line number entries in a common object file

SYNOPSIS

#include <linenum.h>

DESCRIPTION

Compilers based on *pcc* generate an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the −g option; see *cc*(1)). Users can then reference line numbers when using the appropriate software test system (see *sdb*(1)). The structure of these line number entries appears below.

```
struct  lineno
{
        union
        {
                long        l_symndx ;
                long        l_paddr ;
        }                   l_addr ;
        unsigned short      l_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has *l_lnno* equal to zero, and the symbol table index of the function's entry is in *l_symndx*. Otherwise, *l_lnno* is non-zero, and *l_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

l_addr	*l_lnno*
function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0
physical address	line
physical address	line
...	

SEE ALSO

a.out(4), cc(1), sdb(1).

SUPPORT STATUS

Supported.

NAME

master − master device information table

DESCRIPTION

Master is used by *config*(1M) to obtain device information that enables it to generate the configuration files. The file consists of three parts, each separated by a line with a dollar sign ($) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains lines consisting of at least 10 fields and at most 13 fields, with the fields delimited by tabs and/or blanks:

Field 1: device name (8 chars. maximum).

Field 2: interrupt vector size (decimal, in bytes).
- 0 invalid value
- 1-3 yields isr name handler'bd #'int
- 4 yields isr name handlerintr
- 5 yields isr name handler0int
- 8-11 yields isr names handler0int and handler1int
- 12-15 yields isr names handler0int-handler2int
- 16-19 yields isr names handler0int-handler3int
- 20-23 yields isr names handler0int-handler4int
- 24-27 yields isr names handler0int-handler5int
- 28-31 yields isr names handler0int-handler6int
- 32 yields isr names handler0int-handler7int

Field 3: device mask (octal)—each on bit indicates that the handler exists:
- 001000 DMA segments
- 000400 diagnostic handler
- 000200 tty structures
- 000100 initialization handler
- 000040 power-failure handler
- 000020 open handler
- 000010 close handler
- 000004 read handler
- 000002 write handler
- 000001 ioctl handler.

Field 4: device type indicator (octal):
- 000400 nbvl controller
- 000200 allow only one of these devices
- 000100 suppress count field in the **conf.c** file
- 000040 suppress interrupt vector
- 000020 required device

	000010 block device
	000004 character device
	000002 reserved
	000001 reserved
Field 5:	handler prefix (4 chars. maximum).
Field 6:	device address size (decimal).
Field 7:	major device number for block-type device.
Field 8:	major device number for character-type device.
Field 9:	maximum number of devices per controller (decimal).
Field 10:	maximum bus request level (4 through 7).
Fields 11-13:	optional configuration table structure declarations (8 chars. maximum).

Part 2 contains lines with 2 fields each:

Field 1:	alias name of device (8 chars. maximum).
Field 2:	reference name of device (8 chars. maximum; specified in part 1).

Part 3 contains lines with 2 or 3 fields each:

Field 1:	parameter name (as it appears in description file; 20 chars. maximum)
Field 2:	parameter name (as it appears in the **conf.c** file; 20 chars. maximum)
Field 3:	default parameter value (20 chars. maximum; parameter specification is required if this field is omitted)

Devices that are not interrupt-driven have an interrupt vector size of zero. The 040 bit in Field 4 causes *config*(1M) to record the interrupt vector although the **univec.c** file will show no interrupt vector assignment at those locations (interrupts here will be treated as strays).

SEE ALSO

config(1M).

SUPPORT STATUS

Supported.

NAME

mnttab – mounted file system table

SYNOPSIS

#include <mnttab.h>

DESCRIPTION

Mnttab resides in directory /etc and contains a table of devices, mounted by the *mount*(1M) command, in the following structure as defined by **<mnttab.h>**:

```
struct  mnttab {
        char        mt_dev[32];
        char        mt_filsys[32];
        short       mt_ro_flg;
        time_t      mt_time;
};
```

Each entry is 72 bytes long; the first 32 bytes are the null-padded name of the place where the *special file* is mounted; the next 32 bytes represent the null-padded root name of the mounted special file; the remaining 6 bytes contain read/write permissions for the mounted *special file* and the date on which it was mounted.

The maximum number of entries in *mnttab* is based on the system parameter NMOUNT located in the configuration file (i.e. **/kernel/tower/cf/*.cf**), which defines the number of allowable mounted special files.

SEE ALSO

mount(1M), setmnt(1M).

SUPPORT STATUS

Supported.

NAME

passwd − password file

DESCRIPTION

Passwd is an ASCII file which contains the following information for each user:

login name
encrypted password
numerical user ID
numerical group ID
GCOS job number, box number, optional GCOS user ID
initial working directory
program to use as Shell

Each field is separated from the next by a colon.

The GCOS field is used only when communicating with that system; in other installations, the GCOS field can contain any desired information.

Each user is separated from the next by a new-line.

If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user IDs to names.

PASSWORD AGING

The encrypted password consists of 13 characters chosen from a 64 character alphabet (., /, 0−9, A−Z, a−z), except when the password is null in which case the encrypted password is also null. Password aging is effected for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired is forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. A null string is equivalent to zero.

M and *m* have numerical values in the range 0−63 that correspond to the 64 character alphabet shown above (i.e. / = 1 week; z = 63 weeks). If $m = M = 0$ (derived from the string . or ..) the user is forced to change his password the next time he logs in, and the age disappears from his entry in the password file. If $m > M$ (signified, e.g., by the string ./) only the super-user is able to change the password.

FILES

/etc/passwd

SEE ALSO

a64l(3C), crypt(3C), getpwent(3C), group(4), login(1), passwd(1).

SUPPORT STATUS

Supported.

NAME

plot − graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3X) and are interpreted for various devices by commands described in *tplot*(1G).

A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order.

A point is designated by four bytes representing the **x** and **y** values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the current point for the next instruction.

INSTRUCTIONS

Each of the following descriptions is preceded with the name of the corresponding routine in *plot*(3X).

m (move) The next four bytes give a new current point.

n (cont) Draw a line from the current point to the point given by the next four bytes. See *tplot*(1G).

p (point) Plot the point given by the next four bytes.

l (line) Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t (label) Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

e (erase) Start another frame of output.

f (linemod) Take the following string, up to a new-line, as the style for drawing further lines. The styles are *dotted*, *solid*, *longdashed*, *shortdashed*, and *dotdashed*. Effective only for the **−T4014** and **−Tver** options of *tplot*(1G) (TEKTRONIX 4014 terminal and Versatec plotter).

s (space) The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot is magnified or reduced to fit the device as closely as possible.

SPACE SETTINGS

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot*(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	
	space(0, 0, 3120, 3120);

Versatec plotter space(0, 0, 2048, 2048);

SEE ALSO

gps(4), term(5), graph(1G), tplot(1G), plot(3X).

SUPPORT STATUS

Not supported.

NAME

reloc — relocation information for a common object file

SYNOPSIS

#include <reloc.h>
#include <asld.h> (16-bit System Release 2 only)

DESCRIPTION

Unlinked object files contain a relocation entries for each relocatable address reference made in the module. A table of relocation structures is associated with each section of the object module that contains all the information nessesary for *ld(1)* to relocate these references at link time. A section header file pointer and count field points to the first entry and indicates the size of the relocation table for the section.

As the link editor reads each input file and processes the input segments, the relocation table for the segment is read and specific relocation actions are performed on the input data before it is copied to the output object file. The table entries direct how the relocation is done, as they specify the start relocation address, the symbol to be used during relocation, the size of the relocation field, and the type of relocation to perform. Portions of the segment not needing relocation are copied as they are.

Relocation table entries have the following structure:

```
struct reloc
{
   long  r_vaddr ;  /* (virtual) address of reference */
   long  r_symndx ; /* index into symbol table */
   short r_type ;   /* relocation type */
} ;
```

The *r_vaddr* field indicates the position of the data to be modified. This position is relative to the start of the segment, not to the start of the file. Thus, a command to relocate the first byte of the data segment would have an *r_vaddr* field of zero.

The *r_symndx* field is used for undefined external symbols. All undefined external symbols are read into a symbol array, using a C-style index. The value of the field is the index into the symbol array.

The *r_type* field indicates the type of relocation to be done during link editing. Listed below are the relocation types.

In the 16-bit system Release 3 and the 32-bit systems objects, the field is set to one of the following:

```
#define R_ABS       000000 /* Relocation already performed */
#define R_RELBYTE   017    /* A direct 8-bit reference to
                              a symbol's virtual address */
#define R_RELWORD   020    /* A direct 16-bit reference to
                              a symbol's virtual address */
#define R_RELLONG   021    /* A direct 32-bit reference to
                              a symbol's virtual address */
#define R_PCRBYTE   022    /* A "PC-relative" 8-bit reference to
                              a symbol's virtual address */
```

```
#define R_PCRWORD  023    /* A "PC-relative" 16-bit reference to
                             a symbol's virtual address */
#define R_PCRLONG  024    /* A "PC-relative" 32-bit reference to
                             a symbol's virtual address
                             (32-bit systems only) */
```

In 16-bit system Release 2 objects, the field is defined as being made up of a relocation segment id, a relocation size, and a displacement flag. These are added togeter to form *r_rtype* by addition (i.e. relocation segment + relocation size + relocation displacement).

The segment start address is used as the base for the specified relocation segment id. If this is external, the external symbol value is added.

The relocation information is defined as follows in the include file *asld.h*:

```
#define RSEGMNT 0140000 /* Relocation segment field */
#define RTEXT   0000000 /*  - Relocation WRT  text segment       */
#define RDATA   0040000 /*  - Relocation WRT  data segment       */
#define RBSS    0100000 /*  - Relocation WRT  bss  segment       */
#define REXT    0140000 /*  - Relocation WRT  an external symbol  *
#define RSIZE   0030000 /* Relocation size field    */
#define RBYTE   0000000 /*  - 8-bit reference to a symbol address  */
#define RWORD   0010000 /*  - 16-bit reference to a symbol address */
#define RLONG   0020000 /*  - 32-bit reference to a symbol address */
#define RDISP   0004000 /* Relocation dispalcement field (1=>yes) */
```

Relocation entries are generated automatically by the assembler and automatically utilized by the link editor. Link editor options exist for for both preserving and removing the relocation entries for object modules.

SEE ALSO

as(1), ld(1), strip(1), a.out(4), syms(4).

SUPPORT STATUS

Supported.

NAME

rmnttab – mounted directory table

SYNOPSIS

#include <rmnttab.h>

DESCRIPTION

Rmnttab resides in directory /etc and is a table of directories, mounted by the *rmnt*(1M) command, in the following structure defined by **<rmnttab.h>**:

```
#define NRMNT      20
#define DSIZE 64
struct rmnttab {
        char    rmt_dir1[DSIZE],
                rmt_dir2[DSIZE];
        short   rmt_ro_flg;
        time_t  rmt_time;
};
```

Each entry is DSIZE*2+4 bytes in length:

- the first DSIZE bytes are the null-padded pathname of *directory1*, the directory that is rmounted
- the next DSIZE bytes represent the null-padded pathname of *directory2*, the directory that *directory1* is rmounted under
- the remaining 4 bytes contain the date on which it was rmounted.

SEE ALSO

rmnt(1M).

SUPPORT STATUS

Supported.

NAME

sccsfile − format of SCCS file

DESCRIPTION

An SCCS file is an ASCII file. It consists of six logical parts:

checksum	sum of all character values
delta table	information about each delta
user names	login names and/or numerical group IDs of users who may add deltas
flags	definitions of internal keywords
comments	arbitrary descriptive information about the file
body	the actual text lines intermixed with control lines

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and is represented graphically as @. Any line described below which is not depicted as beginning with @ is prevented from beginning with @.

Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @h provides a *magic number* of (octal) 064001.

Delta table

The delta table consists of a variable number of entries of the following forms:

@s *ins/del/unch/*

The number of lines inserted *ins*, deleted *del*, and unchanged *unch*.

@d <type> <SCCS ID> *yr/mo/da hr:mi:se* *<pgmr>* **DDDDD DDDDD**

Descriptive information about the delta:

type	*description*
D	normal
R	removed
SCCS ID	SCCS ID of the delta
yr/mo/da hr:mi:se	date and time of creation of the delta
<pgmr>	login name corresponding to the real user ID at the time the delta was created
DDDDD DDDDD	serial numbers of the delta and its

predecessor, respectively

@i DDDDD ... (optional)
Contain the serial numbers of deltas included.

@x DDDDD ... (optional)
Contain the serial numbers of deltas excluded.

@g DDDDD ... (optional)
Contain the serial numbers of deltas ignored.

@m <MR number>
.
.
.
Each optional line contains one MR number associated with the delta;

@c <comments> ...
.
.
.
Comments associated with the delta.

@e
The end of the delta table entry.

User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags

Keywords used internally (see *admin*(1) for more information on their use). Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:

@f t <type of program>
Defines the replacement for the %Y% identification keyword.

@f v <program name>
Controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program.

@f i
Controls the warning/error aspect of the *No id keywords* message. When the i flag is not present, this message is only a warning; when the i flag is present, this message causes a fatal error (the file is not retrieved, or the delta is not made).

@f b
When the **b** flag is present the **−b** keyletter may be used on the *get* command to cause a branch in the delta tree.

@f m <module name>
Defines the first choice for the replacement text of the %M% identification keyword.

@f f <floor>
Defines the *floor* release (the release below which no deltas may be added).

@f c <ceiling>
Defines the *ceiling* release (the release above which no deltas may be added).

@f d <default-sid>
Defines the default SID to be used when none is specified on a *get* command.

@f n
Causes *delta* to insert a *null* delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty.

@f j
Causes *get* to allow concurrent edits of the same base SID.

@f l <lock-releases>
Defines a *list* of releases that are *locked* against editing (*get*(1) with the −e keyletter).

@f q <user defined>
Defines the replacement for the %Q% identification keyword.

@f z <reserved for use in interfaces>
Used in certain specialized interface programs.

Comments

Arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically contains a description of the purpose of the file.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines:

@I DDDDD insert
@D DDDDD delete
@E DDDDD end

The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1).
Source Code Control System User Guide in the Support Tools Guide.

SUPPORT STATUS

Supported.

NAME

scnhdr – section header for a common object file

SYNOPSIS

#include <scnhdr.h>

DESCRIPTION

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct scnhdr
{
        char            s_name[SYMNMLEN]; /* section name */
        long            s_paddr;    /* physical address */
        long            s_vaddr;    /* virtual address */
        long            s_size;     /* section size */
        long            s_scnptr;   /* file ptr to raw data */
        long            s_relptr;   /* file ptr to relocation */
        long            s_lnnoptr;  /* file ptr to line numbers */
        unsigned short  s_nreloc;   /* # reloc entries */
        unsigned short  s_nlnno;    /* # line number entries */
        long            s_flags;    /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to *fseek*(3S).

If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. An uninitialized section has a size, symbols defined in it, and symbols that refer to it; but it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s_scnptr*, *s_relptr*, *s_lnnoptr*, *s_nreloc*, and *s_nlnno* are zero.

SEE ALSO

fseek(3S), a.out(4), ld(1).

SUPPORT STATUS

Supported.

NAME

syms − common object file symbol table format

SYNOPSIS

#include <syms.h>

DESCRIPTION

Common object files contain information to support *symbolic* software testing (see *sdb*(1)). Line number entries, *linenum*(4), and extensive symbolic information permit testing at the C *source* level. Every object file symbol table is organized as shown below.

```
File name 1.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 1.

File name 2.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 2.
...

Defined global symbols.
Undefined global symbols.
```

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

```
#define SYMNMLEN    8
#define FILNMLEN    14

struct syment
{
   union                                 /* all ways to get symbol name */
   {
      char          _n_name[SYMNMLEN]; /* symbol name */
      struct
      {
         long       _n_zeroes;       /* == 0L when in string table */
         long       _n_offset;       /* location of name in table */
      } _n_n;
      char          *_n_nptr[2];     /* allows overlaying */
   } _n;
   long             n_value;         /* value of symbol */
   short            n_scnum;         /* section number */
   unsigned short   n_type;          /* type and derived type */
   char             n_sclass;        /* storage class */
   char             n_numaux;        /* number of aux entries */
```

```
};

#define n_name     _n._n_name
#define n_zeroes   _n._n_n._n_zeroes
#define n_offset   _n._n_n._n_offset
#define n_nptr     _n._n_nptr[1]
```

Meaningful values and explanations for them are given in both **syms.h** and *Common Object File Format*. Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
        struct
        {
                long            x_tagndx;
                union
                {
                        struct
                        {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union
                {
                        struct
                        {
                                long    x_lnnoptr;
                                long    x_endndx;
                        }       x_fcn;
                        struct
                        {
                                unsigned short  x_dimen[DIMNUM];
                        }       x_ary;
                }               x_fcnary;
                unsigned short  x_tvndx;
        }       x_sym;
        struct
        {
                char    x_fname[FILNMLEN];
        }       x_file;
        struct
        {
                long   x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        }       x_scn;

        struct
```

```
        {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        }       x_tv;
};
```

Indexes of symbol table entries begin at *zero*.

SEE ALSO

a.out(4), linenum(4), sdb(1).
Common Object File Format in the Support Tools Guide.

WARNINGS

On machines in which *long*s are equivalent to *int*s, they are converted to *int*s in the compiler to minimize the complexity of the compiler code generator. Thus the information about which symbols are declared as *long*s and which, as *int*s, does not show up in the symbol table.

SUPPORT STATUS

Supported.

NAME

SYSIDENT – date and release number of the operating system

SYNOPSIS

/etc/SYSIDENT

DESCRIPTION

SYSIDENT is a file in /etc which contains the date and release number of the operating system.

EXAMPLE

050185 RELEASE 030100

SUPPORT STATUS

Supported.

NAME

term − format of compiled term file.

SYNOPSIS

term

DESCRIPTION

Compiled terminfo descriptions are placed under the directory **/usr/lib/terminfo.** In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: **/usr/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/lib/terminfo/a/act4.** Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the *compile* program, and read by the routine *setupterm*. Both of these pieces of software are part of *curses*(3X). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value −1 is represented by 0377, 0377, other negative value are illegal. The −1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the | character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The capabilities are in the same order as the file <term.h>.

Between the boolean section and the number section, a null byte is inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is −1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of −1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information $<nn> and parameter information %x are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for *setupterm* to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since *setupterm* has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine *setupterm* must be prepared for both possibilities — this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
   cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
   ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
   cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],
```

```
000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020   o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040   t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100  \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120 377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140  \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160 377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/*/*   compiled terminal capability data base

SEE ALSO

curses(3X), terminfo(4).

SUPPORT STATUS

Supported.

NAME

terminfo — terminal capability data base

SYNOPSIS

/usr/lib/terminfo/*/*

DESCRIPTION

Terminfo is a data base describing terminals used, for example, by *vi*(1) and *curses*(3X). Terminals are described in *terminfo* by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *terminfo.*

Entries in *terminfo* consist of a number of ',' separated fields. White space after each ',' is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621". This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

CAPABILITIES

The variable is the name by which the programmer (at the terminfo level) accesses the capability. The capname is the short name used in the text of the database, and is used by a person updating the database. The i.code is the two letter internal code used in the compiled database, and always corresponds to the old **termcap** capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file caps to line up nicely. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

(P) indicates that padding may be specified

(G) indicates that the string is passed through tparm withparms as given (#i).

(*) indicates that padding may be based on the number of lines affected

(#i) indicates the *i*th parameter.

Variable Booleans	Cap-name	I. Code	Description
auto_left_margin,	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (e.g.,, dialup, switch).
hard_copy,	hc	hc	Hardcopy terminal
has_meta_key,	km	km	Has a meta key (shift, sets parity bit)
has_status_line,	hs	hs	Has extra "status line"
insert_null_glitch,	in	in	Insert mode distinguishes nulls
memory_above,	da	da	Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode,	mir	mi	Safe to move while in insert mode
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; can not print ~'s
transparent_underline,	ul	ul	underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking
Numbers:			
columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed

virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	No. columns in status line

Strings:

back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Terminal settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter);
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode

exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdl1	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10

lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times. (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes.
reset_2string,	rs2	r2	Reset terminal completely to sane modes.
reset_3string,	rs3	r3	Reset terminal completely to sane modes.
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1

underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non,	mc5p	pO	Turn on the printer for #1 bytes

A Sample Entry

The following entry, which describes the Concept−100, is among the more complex entries in the *terminfo* file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
    cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
    cup=\Ea%p1%' '%+%c%p2%' '%+%c,
    cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
    ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
    il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
    is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
    kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
    kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
    lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
    rev=\ED, rmcup=\Ev    $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
    rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
    smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
    smul=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with "#". Capabilities in *terminfo* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as el (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ','. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..> brackets, as in el=\EK$<3>, and padding characters are supplied by *tputs* to provide this delay. The delay can be either a number, e.g., '20', or a number followed by an '*', i.e., '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of *lines* affected. This is always one unless the terminal has **xenl** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \l \r \t \b \f \s** give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include **\^** for **^**, **** for ****, **\,** for comma, **\:** for **:**, and **\0** for null. (**\0** produces \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a ****.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with *vi* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *terminfo* file to describe it or bugs in *vi*. To easily test a new terminal description you can set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on and programs look there rather than in */usr/lib/terminfo*. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit /etc/passwd at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the 'u' key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as

TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this is a carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over, for example, you would not normally use '**cuf1=** ' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program goes to the bottom left corner of the screen and sends the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge moves to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

```
33 | tty33 | tty | model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM−3 is described as

```
adm3 | 3 | lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf*(3S) like escapes %x in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it. Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often more complex operations are necessary.

The % encodings have the following meanings:

%%	outputs '%'
%d	print pop() as in printf
%2d	print pop() like %2d
%3d	print pop() like %3d
%02d	
%03d	as in printf
%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push ith parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'c'	char constant c
%{nn}	integer constant nn
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%! %~	unary operations push(op pop())
%i	add 1 to first two parms (for ANSI terminals)
%? expr %t thenpart %e elsepart %;	if-then-else, %e elsepart is optional. else-if's are possible ala Algol 68: %? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e %; c_i are conditions, b_i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "%gx%{5}%-".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent preceded by a **^**T, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals which use %c need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n** **^D** and **\r**, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cup=\E=%p1%' '%+%c%p2%' '%+%c. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to **cup** . If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for **home**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **Ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *terminfo*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

Terminfo can describe both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode do not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both are used. The **ich** capability, with one parameter, *n*, repeats the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** affects only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n characters,* and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode*, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking) **bold** (bold or extra bright) **dim** (dim or half-bright) **invis** (blanking or invisible text) **prot** (protected) **rev** (reverse video) **sgr0** (turn off *all* attribute modes) **smacs** (enter alternate character set mode) and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by terminfo.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, ..., f10, the codes they send can be given as **kf0**, **kf1**, ..., **kf10**. If these keys have labels other than the default f0 through f10, the labels can be given as **lf0**, **lf1**, ..., **lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the *tset* command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the *tset* program, each time the user logs in. They are printed in the following order: **is1**; **is2**; setting tabs using **tbc** and **hts**; **if**; running the program **iprog**; and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the *reset* program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the *tset* program to set teletype modes appropriately. Delays embedded in the capabilities **cr, ind, cubl, ff,** and **tab** cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb.**

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a vt100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) is the same as 'xxxxxxxxxx'.

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a CC variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters are not transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software assumes that the 8th bit is parity and it is usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than can fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal is sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Specific Terminal Requirements

Hazeltine terminals, which do not allow '~' characters to be displayed should indicate **hz**.

Terminals which ignore a linefeed immediately after an **am** wrap, such as the Concept and vt100, should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie", that to erase standout mode it is instead necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the escape or control C characters, has **xsb**, indicating that the f1 key is used for escape and f2 for control C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing **xx@** to the left of the capability definition, where xx is the capability. For example, the entry

2621-nl, smkx@, rmkx@, use=2621,

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/usr/lib/terminfo/?/* files containing terminal descriptions

SEE ALSO

curses(3X), printf(3S), term(5), tic(1M).

SUPPORT STATUS

Supported.

NAME

threshold - logalert threshold file

DESCRIPTION

The */ncrm/checklog/threshold* file is read by *logalert* to obtain threshold values. This file is in the */ncrm/checklog* directory, it is owned by *ncrm*, and it has permissions of 644.

The format of this file is structured. Lines beginning with a pound sign (#) are required parameter lines. *Logalert* requires that the order of the parameter lines be fixed: the first parameter line in the file is the error alert message, followed by the bad block message, etc. Lines beginning with a colon (:) are treated as comments. Within a parameter line, white space is treated as a single space.

THRESHOLD FILE PARAMETER LINES

Error alert message

This message is displayed when *logalert* has tallied error records in excess of the threshold value for a device. The message is mailed to the users ncrm, root, and sa and is displayed on the system console. This message can be at most two lines long.

Bad block error message

This message is displayed when *logalert* has tallied error records in excess of the threshold value of bad block errors on a hard disk. The message is mailed to the users ncrm, root, and sa and is displayed on the system console. This message can be at most two lines long.

System console message flag

YES specifies that when errors are detected by *logalert*, the error alert message and bad block message are displayed on the system console. NO suppresses the display of these messages to the console. Note: These messages are always mailed to the users ncrm, root, and sa regardless of the setting of this flag.

Maximum alert{mmddyy} files to retain

Each day *logalert* is run, an *alert{mmddyy}* file is created. This parameter specifies the maximum number of these files to retain in the */ncrm/checklog* directory.

Non-hard disk devices

These parameter lines contain three values per line.

Device name

Device name, for example, f501, tt00, or mem1. The device name all sets the threshold for all devices listed above. The device name ttDF sets the threshold for all terminal devices: pt0a, pt0b, and tt00 through the last terminal device name.

Error threshold

Threshold value, in decimal, for this device. The threshold value of zero causes all errors logged against this device to be ignored.

Transfer flag

Y specifies that if *logalert* has tallied errors for this device exceeding the threshold value, the error records for this device are copied from the error log file to the */ncrm/checklog/alert{mmddyy}* file. N specifies that the transfer of records is suppressed for this device.

Hard disk devices

These parameter lines contain five values per line.

Device Name
: Four character device name, for example, h501, h801, sd01.

Error threshold
: Threshold value, in decimal, for this device. The threshold value of zero causes all errors logged against this device to be ignored.

Error threshold for each block
: When the number of errors detected for an individual block on this device exceeds this threshold value, the bad block error message is displayed.

Maximum blocks with errors
: When the number of blocks with errors on this device exceeds this threshold value, the disk is assumed to have a multitude of problems and the bad block message is not displayed for any of the blocks with errors on this device. The disk controller may have a problem.

Transfer flag
: Y specifies that if *logalert* has tallied errors for this device exceeding the threshold value, the error records for this device are copied from the error log file to the */ncrm/checklog/alert{mmddyy}* file. N specifies that the transfer of records is suppressed for this device.

SEE ALSO

logalert(1M), alert(4).

SUPPORT STATUS

Supported.

NAME

utmp, wtmp – utmp and wtmp entry formats

SYNOPSIS

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

DESCRIPTION

These files, which hold user and accounting information for such commands as *who*(1), *write*(1), and *login*(1), have the following structure as defined by <utmp.h>:

```
#define   UTMP_FILE    "/etc/utmp"
#define   WTMP_FILE    "/etc/wtmp"
#define   ut_name      ut_user

struct utmp {
        char      ut_user[8];         /* User login name */
        char      ut_id[4];           /* /etc/inittab id (usually
                                         line #) */
        char      ut_line[12];        /* device name (console,
                                         lnxx) */
        short     ut_pid;             /* process id */
        short     ut_type;            /* type of entry */
        struct    exit_status {
          short     e_termination;    /* Process termination
                                         status */
          short     e_exit;           /* Process exit status */
        } ut_exit;                    /* The exit status of a
                                         process marked as
                                         DEAD_PROCESS. */

        time_t    ut_time;            /* time entry was made */
};

/* Definitions for ut_type */
#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5                 /* Process spawned by
                                            "init" */
#define LOGIN_PROCESS  6                 /* A "getty" process
                                            waiting for login */
#define USER_PROCESS   7                 /* A user process */
#define DEAD_PROCESS   8
#define ACCOUNTING     9
#define UTMAXTYPE      ACCOUNTING        /* Largest valid value
                                            of ut_type */
```

```
/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length. */
#define RUNLVL_MSG "run-level %c"
#define BOOT_MSG   "system boot"
#define OTIME_MSG  "old time"
#define NTIME_MSG  "new time"
```

FILES

/usr/include/utmp.h
/etc/utmp
/etc/wtmp

SEE ALSO

getut(3C), login(1), who(1), write(1).

SUPPORT STATUS

Supported.

NAME

intro − introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

SUPPORT STATUS

Supported.

NAME

ascii — map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 ´
050 (	051)	052 *	053 +	054 ,	055 –	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fsr	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 ´
28 (	29)	2a *	2b +	2c ,	2d –	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

FILES

/usr/pub/ascii

SUPPORT STATUS

Supported.

NAME

btermcap — terminal capability data base

DESCRIPTION

Btermcap is the Berkeley termcap data base.

FILES

/etc/btermcap

SEE ALSO

termcap(5).

SUPPORT STATUS

Not supported.

NAME

environ — user environment

DESCRIPTION

An array of strings called the *environment* is made available by *exec*(2) when a process begins. By convention, these strings have the form *name=value*. The following names are used by various commands:

PATH
The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1) and others, apply in searching for a file known by an incomplete path name. These prefixes are separated by colons.

Login(1) sets **PATH=:/usr/ucb:/bin:/usr/bin.**

HOME
Name of the login directory of the user, set by *login*(1) from the password file *passwd*(4).

TERM
The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *tplot*(1G), which may exploit special capabilities of that terminal.

TZ
Time zone information. The format is **xxx***n***zzz** where **xxx** is standard local time zone abbreviation, *n* is the difference in hours from GMT, and **zzz** is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT.**

Further names may be placed in the environment by the *export* command and *name=value* arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS.**

SEE ALSO

exec(2), getenv(3C), profile(4), term(5), env(1), login(1), sh(1) mm(1), nice(1), nohup(1), time(1), tplot(1G).

SUPPORT STATUS

Supported.

NAME

eqnchar − special character definitions for eqn and neqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

DESCRIPTION

Eqnchar contains *troff*(1) and *nroff* character definitions for constructing characters that are not available on the Wang Laboratories, Inc. C/A/T phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*.

Eqnchar contains definitions for the following characters:

ciplus	*ciplus*	||	||	*square*	*square*
citimes	*citimes*	*langle*	*langle*	*circle*	*circle*
wig	*wig*	*rangle*	*rangle*	*blot*	*blot*
−wig	*−wig*	*hbar*	*hbar*	*bullet*	*bullet*
>wig	*>wig*	*ppd*	*ppd*	*prop*	*prop*
<wig	*<wig*	<−>	<→	*empty*	*empty*
=wig	*=wig*	<=>	⩽>	*member*	*member*
star	*star*	|<	|<	*nomem*	*nomem*
bigstar	*bigstar*	|>	|>	*cup*	*cup*
=dot	*=dot*	*ang*	*ang*	*cap*	*cap*
orsign	*orsign*	*rang*	*rang*	*incl*	*incl*
andsign	*andsign*	*3dot*	*3dot*	*subset*	*subset*
=del	*=del*	*thf*	*thf*	*supset*	*supset*
oppA	*oppA*	*quarter*	*quarter*	*!subset*	*!subset*
oppE	*oppE*	*3quarter*	*3quarter*	*!supset*	*!supset*
angstrom	*angstrom*	*degree*	*degree*	*scrL*	*scrL*
==<	==<	==>	==>		

FILES

/usr/pub/eqnchar

SEE ALSO

eqn(1), nroff(1), troff(1).

SUPPORT STATUS

Supported.

NAME

fcntl — file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The *fcntl*(2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/* (The first three can only be set by open) */
#define O_RDONLY  0
#define O_WRONLY  1
#define O_RDWR    2
#define O_NDELAY  04       /* Non-blocking I/O */
#define O_APPEND  010      /* append (writes guaranteed
                              at the end) */
#define O_SYNC    020      /* syncronous write option */

/* Flag values accessible only to open(2) */
#define O_CREAT   00400    /* open with file create (uses third
                              open arg)*/
#define O_TRUNC   01000    /* open with truncation */
#define O_EXCL    02000    /* exclusive open */

/* fcntl(2) requests */
#define F_DUPFD   0        /* Duplicate fildes */
#define F_GETFD   1        /* Get fildes flags */
#define F_SETFD   2        /* Set fildes flags */
#define F_GETFL   3        /* Get file flags */
#define F_SETFL   4        /* Set file flags */
#define F_GETLK   5        /* Get blocking file locks */
#define F_SETLK   6        /* Set or clear file locks and
                              fail on busy */
#define F_SETLKW  7        /* Set or clear file locks and
                              wait on busy */
/* file segment locking control structure */
struct flock {
short    l_type;        /* F_RDLCK, F_WRLCK, F_UNLCK */
short    l_whence;      /* flag for starting offset */
long     l_start;       /* reletive offset in bytes */
long     l_len;         /* if zero, then until EOF */
short    l_pid;         /* returned with F_GETLK */
short    l_sysid;       /* unused */
};
/* file segment locking types */
#define F_RDLCK 01      /* Read lock */
#define F_WRLCK 02      /* Write lock */
#define F_UNLCK 03      /* Remove locks */
```

SEE ALSO

fcntl(2), open(2).

SUPPORT STATUS

Supported.

NAME

font − description files for device-independent troff

SYNOPSIS

troff −Tptty ...

DESCRIPTION

For each phototypesetter supported by *troff*(1) and available on this system, there is a directory containing files describing the device and its fonts. This directory is named **/usr/lib/font/dev***ptty* where *ptty* is the name of the phototypesetter. Currently the only *ptty* supported is **aps** for the Autologic APS−5.

For a particular phototypesetter, *ptty*, the ASCII file *DESC* in the directory **/usr/lib/font/dev***ptty* describes its characteristics. Each line starts with a word identifying the characteristic and followed by appropriate specifiers. Blank lines and lines beginning with a # are ignored.

The valid lines for *DESC* are:

res *num*
: resolution of device in basic increments per inch

hor *num*
: smallest unit of horizontal motion

vert *num*
: smallest unit of vertical motion

unitwidth *num*
: pointsize in which widths are specified

sizescale *num*
: scaling for fractional pointsizes

paperwidth *num*
: width of paper in basic increments

paperlength *num*
: length of paper in basic increments

spare1 *num*
: available for use

spare2 *num*
: available for use

sizes *num num ...*
: list of pointsizes available on typesetter

fonts *num name ...*
: number of initial fonts followed by the names of the fonts. For example: fonts 4 R I B S

charset
: this always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

Res is the basic resolution of the device in increments per inch. **Hor** and **vert** describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both **hor** and **vert** would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then **hor** would be 1, while **vert** would be 2. **Unitwidth** is the pointsize in which all width tables in the font description files are given. *Troff* automatically scales the widths from the **unitwidth** size to the pointsize it is working with. **Sizescale** is not currently used and is 1. **Paperwidth** is the width of the paper in basic increments. The APS-5 is 6120 increments wide. **Paperlength** is the length of a sheet of paper in the basic increments.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (e.g., **R**, **I**, **CW**). The format for a font description file is:

name *name*
: name of the font, such as **R** or **CW**

internalname *name*
: internal name of font

special
: sets flag indicating that the font is special

ligatures *name* ... **0**
: Sets flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: **ff fi fl ffi ffl.**

spare1
: available for use

spacewidth *num*
: width of space if something other than 1/3 of \(em is desired as a space.

charset
: The charset must come at the end. Each line following the word *charset* describes one character in the font. Each line has one of two formats:

name width kerningcode
name "

where *name* is either a single ASCII character or a special character name from the list found in *DESC*. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter 'a', and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character.

Troff and its postprocessors read this information from binary files produced from the ASCII files by a program distributed with *troff* called *makedev*. For those with a need to know, a description of the format of these files follows:

The file *DESC.out* starts with the *dev* structure, defined by *dev.h*:

```
/*
dev.h: characteristics of a typesetter
* /

struct dev {
short filesize;  /* number of bytes in file, */
                         /* excluding dev part */
short res;               /* basic resolution in goobies/inch */
short hor;               /* goobies horizontally */
short vert;
short unitwidth;         /* size at which widths are given*/
short nfonts;  /* number fonts physically available */
short nsizes;  /* number of pointsizes */
short sizescale;         /* scaling for fractional pointsizes */
short paperwidth;        /* max line length in units */
short paperlength;  /* max paper length in units */
short nchtab;  /* number of funny names in chtab */
short lchname;           /* length of chname table */
short spare1;  /* in case of expansion */
short spare2;
};
```

Filesize is just the size of everything in *DESC.out* excluding the *dev* structure. *Nfonts* is the number of different font positions available. *Nsizes* is the number of different pointsizes supported by this typesetter. *Nchtab* is the number of special character names. *Lchname* is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansions.

Immediately following the *dev* structure are a number of tables. First is the *sizes* table, which contains *nsizes* + 1 shorts(a null at the end), describing the pointsizes of text available on this device. The second table is the *funny_char_index_table*. It contains indices into the the table which follows it, the *funny_char_strings*. The indices point to the beginning of each special character name which is stored in the *funny_char_strings* table. The *funny_char_strings* table is *lchname* characters long, while the *funny_char_index_table* is *nchtab* shorts long.

Following the *dev* structure will occur *nfonts* {*font*}*.out* files, which are used to initialize the font positions. These {*font*}*.out* files, which also exist as separate files, begin with a *font* structure and then are followed by four character arrays:

```
struct font {   /* characteristics of a font */
char   nwfont; /* number of width entries */
char   specfont;         /* 1 == special font */
char    ligfont;         /* 1 == ligatures exist on this font */
```

```
char   spare1; /* unused for now */
char   namefont[10]; /* name of this font, e.g., R */
char   intname[10]; /* internal name of font, in ASCII */
};
```

The *font* structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font on the typesetting device (*intname*). The internal name is independent of the font position and name that *troff* knows about. For example, you might say mount R in position 4, but when asking the typesetter to actually produce a character from the R font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, *widths*, contains the width of each character relative to *unitwidth*. *Unitwidth* is defined in *DESC*. The second array, *kerning*, contains kerning information. If a character rises above the letter 'a', 02 is set. If it descends below the line, 01 is set. The third array, *codes*, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in *DESC*. It is the *font_index_table*. This table contains indices into the *width*, *kerning*, and *code* tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for *troff* to be able to translate from ASCII and the special character names to these arbitrary tables, the *font_index_table* is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128 - 32, the number of printable characters in the ASCII alphabet. To determine whether a normal ASCII character exists, *troff* takes the ASCII value of the character, subtracts 32, and looks in the *font_index_table*. If it finds a 0, the character is not defined in this font. If it finds anything else, that is the index into *widths*, *kerning*, and *codes* that describe that character.

To look up a special character name, for example **\(pl**, the mathematical plus sign, and determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter'th* position in the *funny_char_index_table*. A string comparision is performed between *funny_char_strings [funny_char_index_table [counter]]* and the special character name, in our example pl, and if it matches, then *troff* refers to this character as (96 + *counter*). When it wants to determine whether a specific font supports this character, it looks in *font_index_table[(96+counter)]*, (see below), to see whether there is a 0, meaning the character does not appear in this font, or number, which is the index into the *widths*, *kerning*, and *codes* tables.

Notice that since a value of 0 in the *font_index_table* indicates that a character does not exist, the 0th element of the *width*, *kerning*, and *codes* arrays are not used. For this reason the 0th element of the *width* array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by *troff* to be 1/3 of the width of the \(em character, but if the 0th element of the *width* array is non-zero, then that value is used for the width of a space.

SEE ALSO

troff(5), troff(1).

FILES

/usr/lib/font/dev{X}/DESC.out description file for phototypesetter X

/usr/lib/font/dev{X}/{font}.out font description files for phototypesetter X

SUPPORT STATUS

Supported.

NAME

greek – graphics for the extended TTY-37 type-box

SYNOPSIS

cat /usr/pub/greek [| greek –Tterminal]

DESCRIPTION

Greek gives the mapping from ASCII to the *shift-out* graphics in effect between SO and SI on TELETYPE® Model 37 terminals equipped with a 128-character type-box. These graphics characters are the default greek characters produced by *nroff*. The filters of *greek*(1) attempt to print them on various other terminals.

The file contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ε	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ϱ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	ϕ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla		[	not		_
partial	∂	]	integral	∫	^			

FILES

/usr/pub/greek

SEE ALSO

300(1), 4014(1), 450(1), greek(1), hp(1), tc(1), nroff(1).

SUPPORT STATUS

Not supported.

NAME

man − macros for formatting entries in this manual

SYNOPSIS

nroff −man files
troff −man [**−rs1**] files

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides standard formatting for the entries of this manual.

OPTIONS

The default page size is 8.5 inches × 11 inches, with a 6.5 inches × 10 inches text area. The **−rs1** option reduces these dimensions to 6 inches × 9 inches and 4.75 inches × 8.375 inches, respectively; this option (which is *not* effective in *nroff*) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point.

The **−rV2** option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining.

MACRO REQUESTS

Any *text* argument below may be one to six *words*. Double quotes may be used to include blanks in a *word*.

If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, .I may be used to italicize a whole line, or .SM followed by .B to make small bold text. By default, hyphenation is turned off for *nroff*, but remains on for *troff*.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., .I, .RB, .SM.

Tab stops are neither used nor set by any macro except .DT and .TH.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*, 5 ens in *nroff*—this corresponds to 0.5 inches in the default page size) by .TH, .P, and .RS, and restored by .RE.

.TH *t s c n* Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary (such as *local*), *n* is new manual name. Invokes .DT (see below).

.SH *text* Place subhead *text*, e.g., SYNOPSIS, here.

.SS *text* Place sub-subhead *text*, e.g., **Options**, here.

.B *text* Make *text* bold.

.I *text* Make *text* italic.

.SM *text* Make *text* 1 point smaller than default point size.

.RI *a b* Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros

alternate between any two of roman, italic, and bold:
.IR .RB .BR .IB .BI

.P	Begin a paragraph with normal font, point size, and indent. **.PP** is a synonym for **.P**.
.HP *in*	Begin paragraph with hanging indent.
.TP *in*	Begin indented paragraph with hanging tag. The next line that contains text to be printed is considered to be the tag. The text following the tag is indented *in* units from the current left margin. If the tag does not fit, it is printed on a separate line.
.IP *t in*	Same as **.TP** *in* with tag *t*; often used to get an indented paragraph without a tag.
.RS *in*	Indent all output an extra *in* units from the current left margin.
.RE *k*	Return to the *k*th relative indent level (initially, $k=1$; $k=0$ is equivalent to $k=1$); if *k* is omitted, return to the most recent lower indent level.
.PM *m*	Produce proprietary markings; where *m* may be **P** for **PRIVATE**, **N** for **NOTICE**, **BP** for **BELL LABORATORIES PROPRIETARY**, or **BR** for **BELL LABORATORIES RESTRICTED**.
.DT	Restore default tab settings (every 7.2 ens in *troff*, 5 ens in *nroff*).
.PD *v*	Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in *troff*, 1v in *nroff*).

The following *strings* are defined:

***R**	® in *troff*, (Reg.) in *nroff*.
***S**	Change to default type size.
***(Tm**	Trademark indicator.

The following *number registers* are given default values by **.TH**:

IN	Left margin indent relative to subheads (default is 7.2 ens in *troff*, 5 ens in *nroff*).
LL	Line length including **IN**.
PD	Current interparagraph distance.

WARNINGS

In addition to the macros, strings, and number registers mentioned above, a number of *internal* macros, strings, and number registers are defined. Except for names predefined by *troff* and number registers **d**, **m**, and **y**, all such internal names are of the form *XA*, where *X* is one of **)**, **]**, and **}**, and *A* is any alphanumeric character.

The macro package increases the inter-word spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font so that one can replace, for example, the bold font by the constant-width font; see *cw*(1). Of course, if the input text of an entry contains requests for other fonts (e.g., .I, .RB, \fI), the corresponding fonts must be mounted during typesetting.

FILES

/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.n.[dt].an
/usr/lib/macros/ucmp.n.an

SEE ALSO

ocw(1), eqn(1), man(1), nroff(1), tbl(1), tc(1), troff(1).

RESTRICTIONS

If the argument to .TH contains *any* blanks and is *not* enclosed by double quotes the output is erroneous.

SUPPORT STATUS

Supported.

NAME

math − math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the *matherr*(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (*e*).
M_LOG2E	The base-2 logarithm of *e*.
M_LOG10E	The base-10 logarithm of *e*.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.)
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the *<values.h>* header file.

FILES

/usr/include/math.h

SEE ALSO

intro(3), matherr(3M), values(5).

SUPPORT STATUS

Supported.

NAME

me − macros for formatting papers

SYNOPSIS

nroff −me [options] file ...
troff −me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a standard formatting facility for technical papers in various formats.

Output of the *eqn*, *neqn*, and *tbl*(1) preprocessors for equations and tables is acceptable as input.

When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined in REQUESTS. Many *nroff* and *troff* requests do not function properly in conjunction with this package, however these requests do function properly after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

REQUESTS

In the following list, initialization refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be: **C** chapter **A** appendix **P** preliminary, e.g., abstract, table of contents, etc.

			B bibliography **RC** chapters renumbered from page one each chapter **RA** appendix renumbered from page one **H** defines the new header. If there are any spaces in it, the entire header must be quoted.
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be: **I** indent equation (default) **L** left-adjust the equation **C** center the equation
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the name of the Author(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augment the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z'*	*''''*	no	Set even footer to x y z
.eh *'x'y'z'*	*''''*	no	Set even header to x y z
.fo *'x'y'z'*	*''''*	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z'*	*''''*	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form .**x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *'x'y'z'*	*''''*	no	Set odd footer to x y z
.oh *'x'y'z'*	*''''*	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.

.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), tbl(1).

SUPPORT STATUS

Not supported.

NAME

mm — the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff −mm [options] [files]

nroff −cm [options] [files]

mmt [options] [files]

troff −mm [options] [files]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **−mm** option causes *nroff*(1) and *troff*(1) to use the non-compacted version of the macro package, while the **−cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

FILES

/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.n.[dt].m	compacted version of the package
/usr/lib/macros/ucmp.n.m	initializers for the compacted version of the package

SEE ALSO

mm(1), mmt(1), nroff(1).

SUPPORT STATUS

Supported.

NAME

mosd – the OSDD adapter macro package for formatting documents

SYNOPSIS

osdd [options] [files]

mm −mosd [options] [files]

nroff −mm −mosd [options] [files]

nroff −cm −mosd [options] [files]

mmt −mosd [options] [files]

troff −mm −mosd [options] [files]

troff −cm −mosd [options] [files]

DESCRIPTION

The OSDD adapter macro package is a tool used in conjunction with the MM macro package to prepare Operations Systems Deliverable Documentation. Many of the OSDD Standards are different than the default format provided by MM. The OSDD adapter package sets the appropriate MM options for automatic production of the OSDD Standards. The OSDD adapter package also generates the correct OSDD page headers and footers, heading styles, Table of Contents format, etc.

Additional Information

OSDD document (input) files are prepared with the MM macros. Additional information which must be given at the beginning of the document file is specified by the following string definitions:

```
.ds H1 document-number
.ds H2 section-number
.ds H3 issue-number
.ds H4 date
.ds H5 rating
```

The *document-number* should be of the standard 10 character format.

The words *Section* and *Issue* should not be included in the string definitions; they are supplied automatically when the document is printed. For example:

```
.ds H1 OPA−1P135−01
.ds H2 4
.ds H3 2
```

automatically produces

```
OPA-1P135-01
Section 4
Issue 2
```

as the document page header. Quotation marks are not used in string definitions.

If certain information is not to be included in a page header, then the string is defined as null; e.g.,

.ds H2

means that there is no *section-number*.

Page Numbering

The OSDD Standards require that certain information such as the document *rating* appear on the *Document Index* or on the *Table of Contents* page if there is no index. By default, it is assumed that an index has been prepared separately. If there is no index, the following must be included in the document file:

.nr Di 0

This ensures that the necessary information is included on the *Table of Contents* page.

The OSDD Standards require that the *Table of Contents* be numbered beginning with *Page 1*. By default, the first page of text will be numbered *Page 2*. If the *Table of Contents* has more than one page, for example *n*, then either **−rP***n+1* must be included as a command line option or **.nr P n** must be included in the document file. For example, if the *Table of Contents* is four pages then use **−rP5** on the command line or **.nr P 4** in the document file.

Numbered Figures

The OSDD Standards require that all numbered figures be placed at the end of the document. The **.Fg** macro is used to produce full page figures. This macro produces a blank page with the appropriate header, footer, and figure caption. Insertion of the actual figure on the page is a manual operation. The macro usage is

.Fg page-count "figure caption"

where *page-count* is the number of pages required for a multi-page figure (default 1 page).

Figure captions are produced by the **.Fg** macro using the **.BS/.BE** macros. Thus the **.BS/.BE** macros are also not available for users. The **.Fg** macro cannot be used within the document unless the final **.Fg** in a series of figures is followed by a **.SK** macro to force out the last figure page.

Table of Contents

The *Table of Contents* for OSDD documents (see Figure 4 in Section 4.1 of the OSDD Standards) is produced with:

.Tc
System Type
System Name
Document Type
.Td

The **.Tc/.Td** macros are used instead of the .TC macro from MM.

NOTICE Disclosure

By default, the adapter package causes the NOTICE disclosure statement to be printed. The .PM macro may be used to suppress the **NOTICE** or to replace it with the **PRIVATE** disclosure statement as follows:

.PM	none printed
.PM P	PRIVATE printed
.PM N	NOTICE printed (default)

Paragraphs

The **.P** macro is used for paragraphs. The **Np** register is set automatically to indicate the paragraph numbering style. It is very important that the **.P** macro be used correctly. All paragraphs (including those immediately following a **.H** macro) must use a **.P** macro. Unless there is a **.P** macro, no number is generated for the paragraph. Similarly, the **.P** macro should not be used for text which is not a paragraph. The **.SP** macro may be appropriate for these cases, e.g., for *paragraphs* within a list item.

Page Header

The page header format is produced automatically in accordance with the OSDD Standards. The OSDD Adapter macro package uses the **.TP** macro for this purpose. Therefore the **.TP** macro normally available in MM is not available for users.

FILES

/usr/lib/tmac/tmac.osd

SEE ALSO

mm(5), mm(1), mmt(1), nroff(1).

SUPPORT STATUS

Supported.

NAME

mptx − the macro package for formatting a permuted index

SYNOPSIS

nroff −mptx [options] [files]

troff −mptx [options] [files]

DESCRIPTION

This package provides a definition for the .xx macro used for formatting a permuted index as produced by *ptx*(1).

This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **−mptx** option must be invoked *after* the **−mm** call. For example:

nroff −cm −mptx file

or

mm −mptx file

FILES

/usr/lib/tmac/tmac.ptx	pointer to the non-compacted version of the package
/usr/lib/macros/ptx	non-compacted version of the package

SEE ALSO

mm(5), mm(1), nroff(1), ptx(1), troff(1).

SUPPORT STATUS

Supported.

NAME

ms − text formatting macros

SYNOPSIS

nroff −ms [options] file ...
troff −ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books.

When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through *col*(1).

All external −ms macros are defined in REQUESTS.

Many *nroff* and *troff* requests function improperly in conjunction with this package. However, the first four requests below work correctly after initialization, and the last two work correctly before initialization:

.bp	begin new page
.br	break output line
.sp n	insert n spacing lines
.ce n	center next n lines
.ls n	line spacing: n=1 single, n=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, \fIword\fR italicizes *word.*

Output of the *tbl, eqn,* and preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

eqn(1), tbl(1), troff(1).

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	−	y	begin abstract; if *x* =no do not label abstract
.AE	−	y	end abstract
.AI	−	y	author's institution
.AM	−	n	better accent mark definitions
.AU	−	y	author's name
.B *x*	−	n	embolden *x*; if no *x*, switch to boldface
.B1	−	y	begin text to be enclosed in a box
.B2	−	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	−	n	print word *x* in a box
.CM	if t	n	cut mark between pages

.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x* =I,L,C,B; *y* =indent
.ID *y*	8n,.5i	y	indented display with no keep; *y* =indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by *eqn*
.EQ *x y*	–	y	break out equation; *x* =L,I,C; *y* =equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y* =indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x* =column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x* =level, *x* =0 resets, *x* =S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x* =no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x* =no stops title

			on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x* =no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x* =H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x* =page or no for none; *y* =indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x* =page or no for none;i *y* =indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Number Registers

Formatting distances can be controlled in −ms by means of built-in number registers. For example

.nr LL 6.5i

sets the line length to 6.5 inches:

The following is a table of number registers and their default values:

Name	*Register Controls*	*Takes Effect*	*Default*
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n

QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, produces output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

String Registers

The following is a list of string registers available in −ms; they may be used anywhere in the text:

Name	*Function of the String*
*Q	quote (" in *nroff*, " in *troff*)
*U	unquote (" in *nroff*, '' in *troff*)
*−	dash (-- in *nroff*, — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

RESTRICTIONS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

SUPPORT STATUS

Not supported.

NAME

mv — troff macro package to typeset view graphs and slides

SYNOPSIS

mvt [**−a**] [options] [files]

troff [**−a**] [**−rX1**] **−mv** [options] [files]

DESCRIPTION

This macro package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *cw*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the Tektronix 4014, as well as on the Versatec printer. For these two devices, specify the **−rX1** option (this option is automatically specified by the *mvt* command−q.v.−when that command is invoked with the **−T4014** or **−Tvp** options). To preview output on other terminals, specify the **−a** option.

The **Tm** string produces the trademark symbol.

The input tilde (~) character is translated into a blank on output.

MACROS

All macros, except those noted, cause a break.

Foil size

The naming convention for the following eight macros is that the first character of the name (**V** or **S**) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**).

The ratio of the longer dimension to the shorter one is larger for slides than for view graphs. As a result, slide foils can be used for view graphs, but not vice versa; on the other hand, view graphs can accommodate a bit more text.

.VS [*n*] [*i*] [*d*] Foil-start macro; foil size is to be 7″×7″; *n* is the foil number, *i* is the foil identification, *d* is the date.

The foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the **.A** macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (**V** or **S**) distinguishes between viewgraphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**). Slides are "skinnier" than the corresponding viewgraphs: the ratio of the longer dimension to the shorter one is larger for slides than for viewgraphs. As a result, slide foils can be used for viewgraphs, but not vice versa; on the other hand, viewgraphs can accommodate a bit more text.

.Vw [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″ wide × 5″ high.
.Vh [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 5″×7″.
.VW [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″×5.4″.
.VH [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″×9″.
.Sw [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″×5″.
.Sh [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 5″×7″.
.SW [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″×5.4″.
.SH [*n*] [*i*] [*d*]	Same as .VS, except that foil size is 7″×9″.

Indentation

.A [*x*]	Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text.
.B [*m* [*s*]]	Place text that follows at the second indentation level preceded by a mark *m* (default is a large bullet). *S* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0). If *s* is 100, the point size of the mark is the same as that of the *default* mark.
.C [*m* [*s*]]	Same as .B, but for the third indentation level; default mark is a dash.
.D [*m* [*s*]]	Same as .B, but for the fourth indentation level; default mark is a small bullet.
.I [*in*] [*a* [*x*]]	Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the .A macro (see below) and passes *x* (if any) to it. The .I macro causes a break only if it is invoked with more than one argument.

Titles

.T *string*	Print *string* as an over-size, centered title.

Typesetting

These macros do not cause a break:

.S [*p*] [*l*]	Set the point size and line length; *p* is the point size (default is previous); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for .VS, .VH, and .SH, and 14 for the other foil-start macros). *l* is the line length (in inches unless dimensioned; default is 4.2″ for .Vh, 3.8″ for .Sh, 5″ for .SH, and 6″ for the other foil-start macros).

.DF *n f* [*n f*...] Define font positions; may not appear within the input text for a foil; it may only appear after all the input text for a foil, but before the next foil-start macro)

n is the position of font *f*; up to four *n f* pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (**H** is a synonym for G):

.DF 1 H 2 I 3 B 4 S

.DV [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for .C, and *d* is for .D; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:

.DV .5v .5v .5v 0v

.U *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

SYNONYMS

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

.AD	.CE	.HY	.NF	.NX	.SP	.TI
.BR	.FI	.NA	.NH	.SO	.TA	

FILES

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

SEE ALSO

cw(1), eqn(1), mmt(1), tbl(1).

RESTRICTIONS

The .VW and .SW foils are meant to be 9″ wide by 7″ high, but because the typesetter paper is generally only 8″ wide, they are printed 7″ wide by 5.4″ high and have to be enlarged by a factor of 9/7 before use as view graphs.

SUPPORT STATUS

Supported.

NAME

prof — profile within a function

SYNOPSIS

#define MARK
#include <prof.h>

void MARK (name)

DESCRIPTION

MARK introduces a mark called *name* that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

Name may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file *<prof.h>* is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc −p −DMARK foo.c
```

If MARK is not defined, the *MARK*(name) statements may be left in the source files containing them and are ignored.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with *MARK* defined on the command line, the marks are ignored.

```
#include <prof.h>

foo( )
{
        int i, j;

        .
        .
        .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

profil(2), monitor(3C), prof(1).

SUPPORT STATUS

Supported.

NAME

regexp — regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include        <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(string, expbuf)
char *string, *expbuf;

extern char *locl, *loc2, *locs;

extern omt circf, sed, nbra;
```

DESCRIPTION

This page describes general purpose regular expression matching routines in the form of *ed*(1), defined in **/usr/include/regexp.h**. Programs such as *ed*(1), *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. Therefore, only this file need be changed to maintain regular expression compatibility.

INIT

Each program that includes **/usr/include/regexp.h** must have a **#define** statement for INIT. This definition is placed right after the declaration for the function *compile* and the opening curly brace ({).

INIT is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise *INIT* can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from *grep*(1).

Macros

Programs that include **/user/include/regexp.h** file must have the following five macros declared before the

include <regexp.h>

statement. These macros are used by the *compile* routine.

GETC()

This macro returns the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.

PEEKC()

This macro returns the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()).

UNGETC(*c*)

This macro forces the next call to GETC() and PEEKC() to return *c*. No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(*c*) is always ignored.

RETURN(*pointer*)

This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which manage memory allocation.

ERROR(*val*)

This macro is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	\digit out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

Compile

SYNTAX

compile(instring, expbuf, endbuf, eof)

ARGUMENTS

Instring is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

Expbuf is a character pointer. It points to the place where the compiled regular expression will be placed.

Endbuf is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf−expbuf*) bytes, a call to

ERROR(50) is made.

Eof is the character which marks the end of the regular expression. For example, in *ed*(1), this character is usually a /.

Step

The function *step* performs actual regular expression matching.

SYNTAX

step(string, expbuf)

ARGUMENTS

String is a pointer to a string of characters to be checked for a match. This string should be null terminated.

Expbuf is the compiled regular expression which was obtained by a call of the function *compile*.

DESCRIPTION

The function *step* returns one, if the given string matches the regular expression, and zero if the expressions do not match.

If there is a match, two external character pointers are set as a side effect to the call to *step*. *Step* sets the external character pointer, *loc1*, to point to the first character that matched the regular expression. The function *advance* sets the variable *loc2*, to point to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* points to the first character of *string* and *loc2* points to the null at the end of *string*.

Step uses the external variable *circf*; *circf* is set by *compile* if the regular expression begins with **^**.

If *circf* is set then *step* only tries to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the first is executed, the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

Advance

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns a one indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called, simply call *advance*.

When *advance* encounters an asterisk (*) or \{ \} sequence in the regular expression it advances its pointer as far as possible in the string and recursively calls itself, trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* backs up in the string, until it finds a match or reaches the point in the string that initially matched the asterisk or \{ \}. Occassionally it is desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* breaks out of the loop that backs up and returns zero.

Locs is used be *ed*(1) and *sed*(1) for substitutions done globally, (not just the first occurrence, but the whole line) so, for example, expressions like s/y*//g do not loop forever.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep*(1):

```
#define INIT            register char *sp = instring;
#define GETC()          (*sp++)
#define PEEKC()         (*sp)
#define UNGETC(c)       (--sp)
#define RETURN(c)       return;
#define ERROR(c)        regerr()

#include <regexp.h>
...
                (void) compile(*argv, expbuf, .expbuf[ESIZE], '\0');
...
                if(step(linebuf, expbuf))
                                succeed();
```

NOTES

The routines *ecmp* and *getrange* and are called by these routines; therefore those names should not be used.

For clarification, look in the source code.

FILES

/usr/include/regexp.h

SEE ALSO

bs(1), ed(1), grep(1), sed(1).

RESTRICTIONS

The routine *ecmp* is equivalent to the Standard I/O routine *strncmp* and should be replaced by that routine.

SUPPORT STATUS

Supported.

NAME

stat – data returned by stat system call

SYNOPSIS

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

DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by these files. The encoding of the field *st.mode* is defined in this file also.

```
/*
 * Structure of the result of stat
 */

struct   stat
{
         dev.t     st.dev;
         ino.t     st.ino;
         ushort    st.mode;
         short     st.nlink;
         ushort    st.uid;
         ushort    st.gid;
         dev.t     st.rdev;
         off.t     st.size;
         time.t    st.atime;
         time.t    st.mtime;
         time.t    st.ctime;
};

#define S.IFMT    0170000 /* type of file */
#define S.IFDIR   0040000 /* directory */
#define S.IFCHR   0020000 /* character special */
#define S.IFBLK   0060000 /* block special */
#define S.IFREG   0100000 /* regular */
#define S.IFIFO   0010000 /* fifo */
#define S.ISUID   04000   /* set user id on execution */
#define S.ISGID   02000   /* set group id on execution */
#define S.ISVTX   01000   /* save swapped text even after use */
#define S.IREAD   00400   /* read permission, owner */
#define S.IWRITE  00200   /* write permission, owner */
#define S.IEXEC   00100   /* execute/search permission, owner */
```

FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

SEE ALSO

stat(2), types(5).

SUPPORT STATUS

Supported.

NAME

term − conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., *nroff*, *mm*(1), *man*(1), *tabs*(1)) and are maintained as part of the shell environment (see *sh*(1), *profile*(4), and *environ*(5)) in the variable $TERM:

1520	Datamedia 1520
1620	Diablo 1620 and others using the HyType II printer
1620−12	same, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631−c	Hewlett-Packard 2631 line printer - compressed mode
2631−e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300−12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s−12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE® Terminal Model 33 KSR
37	TELETYPE Terminal Model 37 KSR
40−2	TELETYPE Terminal Model 40/2
40−4	TELETYPE Terminal Model 40/4
4540	TELETYPE Terminal Model 4540
3270	IBM Model 3270
4000a	Trendata 4000a
4014	Tektronix 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450−12	same, in 12-pitch mode
735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
sync	generic name for synchronous TELETYPE 4540-compatible terminals
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300

Up to 8 characters, chosen from [−a−z0−9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a −. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form —T*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable $TERM, which, in turn, should contain *term*.

SEE ALSO

profile(4), environ(5), man(1), mm(1), nroff(1), tplot(1G), sh(1), stty(1), tabs(1).

SUPPORT STATUS

Supported.

NAME

termcap – terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base of terminal descriptions used by programs such as *vi*(1). Each terminal description consists of a set of capabilities of that terminal, a description of how operations are performed, padding requirements, and the initialization sequence.

Entries in *termcap* consist of a number of fields separated by colons (:). The first entry for each terminal gives the known names for the terminal, separated by | characters.

The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and should contain no blanks.

The last name given should be a long name fully identifying the terminal. It may contain blanks for readability.

The following entry shows the format for a typical description ID field:

nl | 7900 | NCR 7900 Mode 1: ... Description ... :

The description field itself consists of capability fields separated by colon delimiters.

CAPABILITIES

Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not **^H**
bs	bool		Terminal can backspace with **^H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal can be set
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like **cm** but horizontal motion only, line stays same
cl	str	(P*)	Clear screen

cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like **cm**
cv	str	(P)	Like **ch** but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of **bs** delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of **cr** delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of **ff** delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of **nl** delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if **ic**
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give :im=: if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by other function keys 0-9
kb	str		Sent by backspace key
kc	str		Sent by line temination sequence (e.g. NEWLINE)
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		*Termcap* entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by *vi* version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).

nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but does not scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank characters left by **so** or *se*
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one character and move past it
ue	str		End underscore mode
ug	num		Number of blank characters left by **us** or **ue**
ul	bool		Terminal underlines even though it does not overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so character (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept−100, is among the more complex entries in the *termcap* file. This particular entry is outdated, and is used as an example only.

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\
 200\Eo&\200:\
 :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\
  Ea%+ %+ :\
 :co#80:\:dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:\
 :ip=16*:li#24:mi:nd=\E=:\
 :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next).

Special Capabilities

The following capabilities are required to support screen functionally of the named application packages available:

Name	Type	Pad?	Description
NM	str		Cobol runtime : change to normal intensity
NB	str		Cobol runtime : normal blinking
NR	str		Cobol runtime : normal reverse video
NS	str		Cobol runtime : normal blinking and reverse
AL	str		Cobol runtime : change to alternate intensity
AB	str		Cobol runtime : alternate blinking
AR	str		Cobol runtime : normal reverse video
AS	str		Cobol runtime : alternate blinking and reverse
OV	num		Cobol runtime : overhead, same as *ug* and *sg*
CF	str		Cobol runtime : make cursor invisable
CN	str		Cobol runtime : make cursor visable
MP	str		Multiplan : terminal initialization on entry
MR	str		Multiplan : terminal reset on exit
NU	str		Multiplan : keystroke to move to next unlocked cell
EN	str		Multiplan : keystroke to move to end of spread sheet
kA-kZ	str		SNA communication
lA-lZ	str		SNA communication

The SNA capabilities are:

Name	Function Code	Hex Code	Function
kA	FKA	9A	PF11
kB	FKB	9B	PF12
kC	FKC	9C	PF13
kD	FKD	9D	PF14
kE	FKE	9E	PF15
kF	FKF	9F	PF16
kG	FKG	A0	PF17
kH	FKH	A1	PF18
kI	FKI	A2	PF19
kJ	FKJ	A3	PF20
kK	FKK	A4	PF21
kL	FKL	A5	PF22
kM	FKM	A6	PF23
kN	FKN	A7	PF24
kO	FKO	A8	Not used
kP	FKP	A9	Not used
kQ	FKQ	AA	Not used
kR	FKR	AB	Not used
kS	FKS	AC	Not used
kT	FKT	AD	Not used
kU	FKU	AE	PF1
kV	FKV	AF	PF2
kW	FKW	B0	PF3
kX	FKX	B1	PF4
kY	FKY	B2	PF5
kZ	FKZ	B3	PF6
lA	FLA	BE	PF7

lB	FLB	BF	PF8
lC	FLC	C0	PF9
lD	FLD	C1	PF10
lE	FLE	C2	Enter
lF	FLF	C3	PA1
lG	FLG	C4	PA2
lH	FLH	C5	PA3
lI	FLI	C6	Clear
lJ	FLJ	C7	Up; should agree with ku
lK	FLK	C8	Backspace; sequence generated by up arrow cursor key
lL	FLL	C9	Newline
lM	FLM	CA	Tab
lN	FLN	CB	Backtab
lO	FLO	CC	Shiftlock; define only if shiftlock generates a sequence
lP	FLP	CD	Insert
lQ	FLQ	CE	Field mark (FM)
lR	FLR	CF	Attention (ATTN)
lS	FLS	D0	SYS REQ
lT	FLT	D1	ERASE INPUT
lU	FLU	D2	ERASE EOF
lV	FLV	D3	RESET
lW	FLW	D4	DUP
lX	FLX	D5	QUIT
lY	FLY	D6	Hardcopy
lZ	FLZ	D7	DELETE

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has automatic margins (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**.

Numeric capabilities are followed by the character *#* and then the value. Thus **co** which indicates the number of columns the terminal has gives the value *80* for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an =, and then a string ending at the next following colon.

A delay in milliseconds may appear after the = in such a capability, and padding characters are supplied by the software after the remainder of the string is sent to provide this delay. The delay can be either a integer, such as *20*, or an integer followed by an asterisk, such as 3*.

An asterisk indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When an asterisk is specified, it is sometimes useful to specify a delay per unit to tenths of milliseconds, such as 3.5.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters:

Character	*Escape Sequence*
ESCAPE character	\E
control-*x*	^x (for any appropriate *x*)
newline	\n
return	\r
tab	\t
backspace	\b
formfeed	\f
any character	\ddd (where *ddd* are octal digits)
carat	\^
backslash	\\
colon	\072
null character	\200

The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a \200 comes out as a \000 would.

PREPARING DESCRIPTIONS

The most effective way to prepare a terminal description is to imitate the description of a similar terminal in *termcap*, building up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose bugs in *ex* or deficiencies in the ability of the *termcap* file to describe that terminal.

To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the software looks there rather than in */etc/termcap.* TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the software.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability.

If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability.

If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability.

If the terminal can clear its screen, then this is given by the **cl** string capability.

If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than ^H. In that case you should give this character as the **bc** string capability.

If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

NOTE: the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor never attempts to backspace around the left edge, nor does it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen causes the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

t3 | 33 | tty33:co#72:os

while the Lear Siegler ADM−3 is described as

cl | adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability, with escapes like %x in it, similar to those of *printf*(3S). These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output
%D	Reverse coding (x-2*(x%16)), no output (Delta Data)

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is cm=6\E&%r%2c%2Y.

The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, cm=^T%.%..

Terminals which use %. need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit \t, **\n** **^D** and \r, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space).

If it can move the cursor up a line on the screen in the same column, this should be given as **up**.

If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor never does this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**.

If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line.

If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted.

If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices.

If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to the insert/delete character which can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly.

Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks.

You can determine which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type **abc** **def** using local cursor motions (not spaces) between the **abc** and the **def**. Then position the cursor before the **abc** and put the terminal in insert mode.

If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions.

If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for *insert null.*

If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line.

Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position.

Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im**).

Give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode do not give **ic**; terminals which send a sequence to open a screen position should give it here.

Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.

If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** affects only speed. Some terminals, notably Datamedias, must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several kinds of standout mode (such as inverse video, blinking, or underlining – half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself.

If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray

1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively.

If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor.

If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**.

If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys).

If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively.

If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9.**

If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9.**

If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, :ko=cl,ll,sf,sb:, which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, then this can be given as **ta**.

Hazeltine terminals, which do not allow tilde (~) characters to be printed should indicate **hz**.

Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**.

Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**.

If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**.

Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024.

Since *termlib* routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with **xx@** where xx is the capability. For example, the entry

    hn | 2621nl:ks@:ke@:tc=2621:

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap	file containing terminal descriptions
/etc/ttytype	file containing default login terminal descriptions

SEE ALSO

curses(3), termcap(3), ex(1), more(1), tset(1), ul(1), vi(1).

WARNINGS AND RESTRICTIONS

Ex allows only 256 characters for string capabilities, and the routines in *termcap*(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, **ve**, **al**, **dl**, **up**, **do**, **le**, and **ri** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

SUPPORT STATUS

Supported.

NAME

troff – description of output language

DESCRIPTION

The device-independent *troff* outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted as *n* and are ASCII strings. Strings inside of [ ] are optional. *Troff* may produce them, but they are not required for the specification of the language. The character **\n** has the standard meaning of "newline" character. Between commands white space has no meaning. White space characters are spaces and newlines. All commands which have an arbitary length numerical parameter or word must be followed by white space. For example, the command to specify point size, **s**###, must be followed by a space or newline.

s*n*
: The point size of the characters to be generated.

f*n*
: The font mounted in the specified position is to be used. The number ranges from 0 to the highest font presently mounted. 0 is a special position, invoked by *troff*, but not directly accessible to the troff user. Normally fonts are mounted starting at position 1.

c*x*
: Generate the character *x* at the current location on the page; *x* is a single ASCII character.

C*xyz*
: Generate the special character *xyz*. The name of the character is delimited by white space. The name is one of the special characters legal for the typesetting device as specified by the device specification found in the file *DESC*. This file resides in a directory specific for the typesetting device. (See *font*(5) and **/usr/lib/font/dev*.**)

H*n*
: Change the horizonal position on the page to the number specified. The number is in basic units of motions as specified by *DESC*. This is an absolute "goto".

h*n*
: Add the number specified to the current horizontal position. This is a relative "goto".

V*n*
: Change the vertical position on the page to the number specified (down is positive).

v*n*
: Add the number specified to the current vertical position.

nnx
: This is a two-digit number followed by an ASCII character. The meaning is a combination of **h***n* followed by **c***x*. The two digits *nn* are added to the current horizontal position

and then the ASCII character, *x*, is produced. This is the most common form of character specification.

n*b a* — This command indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. *Troff* specifies a resetting of the *x*,*y* coordinates on the page before requesting that more characters be printed. The first number, *b*, is the amount of space before the line and the second number, *a*, the amount of space after the line. The second number is delimited by white space.

w — A w appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device.

p*n* — Begin a new page. The new page number is included in this command. The vertical position on the page should be set to 0.

{ — Push the current environment, which means saving the current point size, font, and location on the page.

} — Pop a saved environment.

t*xxxxx* — Print the string of characters, *xxxxx*, using the natural width of each character to determine the next x coordinate. *Troff* does not currently produce this form of command. It is not recommended. The characters might be too close together.

 # .... \n — A line beginning with a pound sign is a comment.

Dl *x y***\n** — Draw a line from the current location to *x*,*y*. At the end of the drawing operation the current location is *x*,*y*.

Dc *d***\n** — Draw a circle of diameter *d* with the leftmost edge being at the current location (x, y). The current location after drawing the circle is x+*d*,y, the rightmost edge of the circle.

De *dx dy***\n** — Draw an ellipse with the specified axes. *dx* is the axis in the x direction and *dy* is the axis in the y direction. The leftmost edge of the ellipse is at the current location. After drawing the ellipse the current location is x+*dx*,y.

Da *x y r***\n** — Draw a counterclockwise arc from the current location to *x*,*y* using a circle of radius *r* . The current location after drawing the arc is *x*,*y*.

D~ *x y x y*...**\n** — Draw a spline curve (wiggly line) between each of the *x*,*y* coordinate pairs starting at the current location. The final location is the final *x*,*y* pair of the list. Currently there may be no more than 36 *x*,*y* pairs to this command.

x i[nit]\n — Initialize the typesetting device. The actions required are dependent on the device. An **init** command occurs before any output generation is attempted.

x T *device*\n — The name of the typesetter is *device*. This is the same as the argument to the **−T** option. The information about the typesetter is found in the directory **/usr/lib/font/dev**{*device*}.

x r[es] *n h v*\n — The resolution of the typesetting device in increments per inch is *n*. Motion in the horizontal direction can take place in units of *h* basic increments. Motion in the vertical direction can take place in units of *v* basic increments. For example, the APS-5 typesetter has a basic resolution of 723 increments per inch and can move in either direction in 723rds of an inch. Its specification is:
x res 723 1 1

x p[ause]\n — Pause. Cause the current page to finish but do not relinquish the typesetter.

x s[top]\n — Stop. Cause the current page to finish and then relinquish the typesetter. Perform any shutdown and bookkeeping procedures required.

x t[railer]\n — Generate a trailer. On some devices no operation is performed.

x f[ont] *n name*\n — Load the font *name* into position *n*.

x H[eight] *n*\n — Set the character height to *n* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter.

x S[lant] *n*\n — Set the slant to *n* degrees. Only some typesetters can do this and not all angles are supported.

SUPPORT STATUS

Supported.

NAME

types − primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
typedef struct { int r[1]; } *          physadr;
typedef long            daddr_t;
typedef char *          caddr_t;
typedef unsigned int    uint;
typedef unsigned short  ushort;
typedef ushort          ino_t;
typedef short           cnt_t;
typedef long            time_t;
typedef int             label_t[13];
typedef short           dev_t;
typedef long            off_t;
typedef long            paddr_t;
typedef long            key_t;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(4).

Times are encoded in seconds; 00:00:00 GMT is January 1, 1970.

The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent.

Offsets are measured in bytes from the beginning of a file.

The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(4).

SUPPORT STATUS

Supported.

NAME

values — machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*) The number of bits in a specified type (e.g., int).

HIBITS The value of a short integer with only the high-order bit set (in most implementations, 0x8000).

HIBITL The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).

HIBITI The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).

MAXSHORT The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).

MAXLONG The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).

MAXINT The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).

MAXFLOAT, LN_MAXFLOAT The maximum value of a single-precision floating-point number, and its natural logarithm.

MAXDOUBLE, LN_MAXDOUBLE The maximum value of a double-precision floating-point number, and its natural logarithm.

MINFLOAT, LN_MINFLOAT The minimum positive value of a single-precision floating-point number, and its natural logarithm.

MINDOUBLE, LN_MINDOUBLE The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF The number of significant bits in the mantissa of a single-precision floatingpoint number.

DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

FILES

/usr/include/values.h

SEE ALSO

intro(3), math(5).

SUPPORT STATUS

Supported.

NAME

varargs − handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type **va_arg(pvar,** *type***)**
va_list pvar;

void va_end(pvar)
va_list pvar;

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as *printf*(3S)) but do not use *varargs* are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for *va_alist*. No semicolon should follow *va_dcl*.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list.

va_arg returns the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by *va_start* ... *va_end*, are possible.

EXAMPLE

This example is a possible implementation of *execl*(2).

```
#include <varargs.h>
#define MAXARGS    100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS]:
        int argno = 0;
```

```
        va_start(ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
                ;
        va_end(ap);
        return execv(file, args);
}
```

SEE ALSO

exec(2), printf(3S).

RESTRICTIONS

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, *execl* is passed a zero pointer to signal the end of the list. *Printf* can tell how many arguments are there by the format.

It is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, since arguments seen by the called function are not *char*, *short*, or *float*. C converts *char* and *short* arguments to *int* and converts *float* arguments to *double* before passing them to a function.

SUPPORT STATUS

Supported.

SEE REVERSE SIDE OF THIS FORM FOR INSTRUCTIONS

DOCUMENTATION ORDER FORM

NCR Customer No

Purchase Order No	Purchase Order Date

SHIP TO:	Ship To No.:
Company Name	
Address	
City State Zip	
Attention	
In case we have questions regarding your order *Area Code ( ) Number — Ext.*	
☐ Change of Address	

BILL TO:	Bill To No.:
Company Name	
Address	
City State Zip	
Attention	
In case we have questions regarding your order *Area Code ( ) Number — Ext*	
☐ Change of Address	

Ship VIA

International Only — Customs Declaration

Comments

DOCUMENT NUMBER	QUANTITY	DESCRIPTION	UNIT PRICE	AMOUNT
			Subtotal	
			State/Local Taxes	
			Total	

Tax Exempt? ☐ No ☐ Yes	Tax Exempt No

For your convenience you can:

1. Mail this order to:
   NCR Corporation
   Order Processing
   Publication Services
   Dayton, Ohio 45479
2. Submit this order to your local NCR office
3. Call our toll-free number:
   1-800-543-2010
   In Ohio:
   1-800-543-6691
   8:00 A.M.-4:30 P.M.
   EST - Weekdays

NCR

Signature ____________________ Date ________

ORDERING INFORMATION

HOW TO ORDER

For a fast, easy way to order and receive the documents you need, complete this Documentation Order Form as follows.

1. NCR CUSTOMER NO.

Enter your 8-digit NCR customer number.

> ***CUSTOMER NUMBERS ARE REQUIRED ON ALL ORDERS.***
>
> *If you are unsure of your customer number, please contact your local NCR office.*

2. PURCHASE ORDER NO.

Enter your purchase order number.

3. PURCHASE ORDER DATE

Enter the purchase order date.

4. SHIP TO

a. If you wish to have documents shipped to a location different than the location associated with your NCR customer number, enter the appropriate information in the space provided.

b. If the order is to be shipped to your NCR customer number location, no entry is required in this space.

5. BILL TO

No entry is required unless the BILL TO location is different from the SHIP TO location.

6. SHIP VIA

Enter your preferred method of delivery. All orders for items in stock will be processed and shipped within one week of receipt of order via UPS or mail for domestic shipment and air shipment for international orders. Rush orders will be processed and delivered within 48 hours.

7. CUSTOMS DECLARATION

(For International Orders)

If a customs declaration is required, enter the full text of the declaration in the space provided.

8. COMMENTS

Use this space for special comments about your order.

9. DOCUMENT NUMBER

Enter the number of the document you wish to order. NCR document numbers have a 2-character prefix, followed by 4 to 7 characters. (D1-0000-00).

10. QUANTITY

Enter the quantity desired. The following discounts apply when any document is purchased in quantities of 10 or more on one order.

- 10-49 10% discount
- 50-99 15% discount
- 100 or more 20% discount

11. DESCRIPTION

Enter the title of the document.

12. UNIT PRICE

Enter the unit price of the document. (Prices listed are those in effect at the time of printing and are subject to change without notice.) All prices are quoted in U.S. dollars.

13. AMOUNT

To calculate the line item amount, multiply the quantity by the unit price.

14. SUBTOTAL

Add all entries in the amount column and enter the sum in the SUBTOTAL space.

15. STATE/LOCAL TAXES

Calculate applicable state and local taxes by multiplying the SUBTOTAL amount by the percentage of tax. Enter that number in the STATE/LOCAL TAXES space. If tax exempt, your tax exempt number must be entered in the space provided to the left.

16. TOTAL

Enter the total amount of the order.

THREE CONVENIENT WAYS TO ORDER

Enter your order using one of the following convenient methods:

U.S. Customers

- Mail the Documentation Order Form to NCR Corporation, Order Processing—Publication Services, Dayton, Ohio 45479.
- Submit the order to your local NCR office.
- Call our toll-free number: **1-800-543-2010**; in Ohio, **1-800-543-6691**. Phone orders are taken from 8:00 am to 4:30 pm EST—weekdays.

International Customers

- Customers located outside of the United States should contact their local NCR office for ordering and pricing information.

PAYMENT METHODS

Payments are accepted by check, money order, or purchase order. Please do not send cash.

RETURNS

If you wish to return documents for credit, please contact our Order Processing department (1-800-543-2010; in Ohio, 1-800-543-6691) within 30 days of the shipment date. Full credit will be given if documents are returned because of an NCR error. A credit of up to 75% of the net amount may be issued for the return of unused, shrink-wrapped documents. Returns will NOT be accepted without prior approval from Publication Services.

OUT OF STOCK ITEMS

Every attempt will be made to ensure that your order is filled completely and accurately. If a document is temporarily out of stock, it will be placed on back order. When a partial shipment is made, your packing list will include a notification of this condition. Back orders will be automatically filled when the document is returned to our inventory.

Retain a copy of this order for your records. Thank you

NCR

READER'S COMMENTS FORM

x 2066 90 0884

BOOK TITLE	BOOK NO.	PRINT DATE

To help us plan future editions of this document, please take a few minutes to answer the following questions. Explain in detail using the space provided. Include page numbers where applicable.

Are there any technical errors or misrepresentations in the document?

Is the material presented in a logical and consistent order?

Is it easy to locate specific information in the document?

Is there any information you would like to have added to the document?

Are the examples relevant to the task being described?

Could parts of the document be deleted without affecting the document's usefulness?

Did the document help you to perform your job?

Any general comments?

NAME ______________________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

TELEPHONE NO ( ) ______________________

Thank you for your evaluation of this document

Fold the form as indicated and mail to NCR No postage is necessary in the U S A

TAPE

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.3 DAYTON, OHIO

POSTAGE WILL BE PAID BY ADDRESSEE

NCR Corporation

ATTENTION: Publication Services
WHQ-4

Dayton, Ohio 45409

UPDATE NOTIFICATION

Document No.: D1-0399-C

Date of Update: October, 1986

Document Title: TOWER Programmer Reference Manual

Original Issue Date: November, 1985

REASON FOR UPDATE:

To include new release of software.

UPDATE INSTRUCTIONS:

The enclosed book is for TOWER 32 Operating System Release 1.03 and TOWER XP Operating System Release 3.01

If you have TOWER 32 Operating System Release 1.02, keep your current book until you upgrade to Release 1.03.

If you have TOWER 32 Operating System Release 1.02 but do not have the Release 1.02 book, this book should be used. When using this book, be aware that the following manual pages only pertain to Release 3.01:

- sernum .... get serial number of current system
- lockf .... record locking on files
- alert .... error records for devices exceeding threshold values
- alertmesg .... logalert summary message file
- queuedefs .... cron and at queue definition file
- threshold .... threshold - logalert threshold file

If you have TOWER XP Operating System Release 3.01, replace the old book in its entirety with this new book.

If you have TOWER XP Operating System Release 3.00, keep your Release 3.00 book.

NCR