```
  /$$$$$$   /$$    /$$   /$$$$$$$   /$$$$$$  /$$$$$$$
 /$$__  $$ /$$$$  /$$$$ | $$__  $$ /$$__  $$| $$__  $$
|__/  \ $$|_  $$ |_  $$ | $$  \ $$| $$  \__/| $$  \ $$
  /$$$$$$/  | $$   | $$ | $$$$$$$ |  $$$$$$ | $$  | $$
 /$$____/   | $$   | $$ | $$__  $$ \____  $$| $$  | $$
| $$        | $$   | $$ | $$  \ $$ /$$  \ $$| $$  | $$
| $$$$$$$$ /$$$$$$ /$$$$$$| $$$$$$$/|  $$$$$$/| $$$$$$$/
|________/|______/|______/|_______/  \______/ |_______/

*******************************************************
************** The 211BSD man page project *************
*******************************************************
*********** Manual 3 - C Library Subroutines ***********
*******************************************************
```

```
Inspired by:
===============================================================================
  SimH      http://simh.trailing-edge.com/
  PiDP11    https://obsolescence.wixsite.com/obsolescence/pidp-11
  BSD 2.11  https://wfjm.github.io/home/211bsd/
```

```
Presented by the ShadowTron Blog
===============================================================================
  https://www.youtube.com/c/shadowtronblog
  www.shadowtron.com
  shadowtronblog <at> gmail <dot> com
```

```
Other manuals in the series
===============================================================================
    Manual 1  - Commands and Application Programs
    Manual 2  - System Calls
==> Manual 3  - C Library Subroutines
    Manual 3F - Fortran Library
    Manual 4  - Special Files
    Manual 5  - File Formats
    Manual 6  - Games
    Manual 7  - Miscellaneous
    Manual 8  - System Maintenance
```

Version 3.0

```
*****************************************
**** Manual 3 - C Library Subroutines ****
*****************************************
```

NAME

intro - introduction to C library functions

DESCRIPTION

This section describes functions that may be found in various libraries. The library functions are those other than the functions which directly invoke UNIX system primitives, described in section 2. Most of these functions are accessible from the C library, libc, which is automatically loaded by the C compiler cc(1), and the Pascal compiler pc(1). The link editor ld(1) searches this library under the `-lc' option. The C library also includes all the functions described in section 2.

A subset of these functions are available from Fortran; they are described separately in intro(3F).

The functions described in this section are grouped into various sections:

(3) The straight ``3'' functions are the standard C library functions.

(3N) These functions constitute the internet network library.

(3S) These functions constitute the `standard I/O package', see stdio(3S) for more details. Declarations for these functions may be obtained from the include file <stdio.h>.

(3C) These routines are included for compatibility with other systems. In particular, a number of system call interfaces provided in previous releases of 4BSD have been included for source code compatibility. Use of these routines should, for the most part, be avoided. The manual page entry for each compatibility routine indicates the proper interface to use.

(3M) These functions constitute the math library, libm. When functions in the math library (see math(3M)) are passed values that are undefined or would generate answers that are out of range, they call the infnan routine. By default this routine returns the VAX reserved floating point value which causes the process to get a floating point exception (see sigvec(2)). Programs that wish to take other action should define their own version of infnan (see infnan(3M) for details). The math library is loaded as needed by the Pascal compiler pc(1). C programs that wish to use this library need to specify the ``-lm'' option.

(3X) These functions constitute minor libraries and other miscellaneous run-time facilities. Most are available only when programming in C. These functions include libraries that provide device independent plotting functions, terminal independent screen management routines for two dimensional non-bitmap display terminals, and functions for managing data bases with inverted indexes. These functions are located in separate libraries indicated in each manual entry.

FILES

/lib/libc.a	the C library
/usr/lib/libm.a	the math library
/usr/lib/libc_p.a	the C library compiled for profiling
/usr/lib/libm_p.a	the math library compiled for profiling

SEE ALSO

stdio(3S), math(3M), intro(2), cc(1), ld(1), nm(1)

LIST OF FUNCTIONS

Name	Appears on Page	Description
abort	abort.3	generate a fault
abs	abs.3	integer absolute value
acos	sin.3m	inverse trigonometric function
acosh	asinh.3m	inverse hyperbolic function
alarm	alarm.3c	schedule signal after specified time
alloca	malloc.3	memory allocator
arc	plot.3x	graphics interface
asctime	ctime.3	convert date and time to ASCII
asin	sin.3m	inverse trigonometric function
asinh	asinh.3m	inverse hyperbolic function
assert	assert.3x	program verification
atan	sin.3m	inverse trigonometric function
atanh	asinh.3m	inverse hyperbolic function
atan2	sin.3m	inverse trigonometric function
atof	atof.3	convert ASCII to numbers
atoi	atof.3	convert ASCII to numbers
atol	atof.3	convert ASCII to numbers
bcmp	bstring.3	bit and byte string operations
bcopy	bstring.3	bit and byte string operations
bzero	bstring.3	bit and byte string operations
cabs	hypot.3m	complex absolute value
calloc	malloc.3	memory allocator
cbrt	sqrt.3m	cube root
ceil	floor.3m	integer no less than
circle	plot.3x	graphics interface
clearerr	ferror.3s	stream status inquiries
closedir	directory.3	directory operations
closelog	syslog.3	control system log
closepl	plot.3x	graphics interface
cont	plot.3x	graphics interface
copysign	ieee.3m	copy sign bit

cos	sin.3m	trigonometric function
cosh	sinh.3m	hyperbolic function
crypt	crypt.3	DES encryption
ctime	ctime.3	convert date and time to ASCII
curses	curses.3x	screen functions with ``optimal'' cursor motion
dbminit	dbm.3x	data base subroutines
delete	dbm.3x	data base subroutines
drem	ieee.3m	remainder
ecvt	ecvt.3	output conversion
edata	end.3	last locations in program
encrypt	crypt.3	DES encryption
end	end.3	last locations in program
endfsent	getfsent.3x	get file system descriptor file entry
endgrent	getgrent.3	get group file entry
endhostent	gethostbyname.3n	get network host entry
endnetent	getnetent.3n	get network entry
endprotoent	getprotoent.3n	get protocol entry
endpwent	getpwent.3	get password file entry
endservent	getservent.3n	get service entry
environ	execl.3	execute a file
erase	plot.3x	graphics interface
erf	erf.3m	error function
erfc	erf.3m	complementary error function
etext	end.3	last locations in program
exec	execl.3	execute a file
exece	execl.3	execute a file
execl	execl.3	execute a file
execle	execl.3	execute a file
execlp	execl.3	execute a file
exect	execl.3	execute a file
execv	execl.3	execute a file
execvp	execl.3	execute a file
exit	exit.3	terminate a process after flushing any pending output
exp	exp.3m	exponential
expm1	exp.3m	exp(x)-1
fabs	floor.3m	absolute value
fclose	fclose.3s	close or flush a stream
fcvt	ecvt.3	output conversion
feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fetch	dbm.3x	data base subroutines
fflush	fclose.3s	close or flush a stream
ffs	bstring.3	bit and byte string operations
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
firstkey	dbm.3x	data base subroutines
floor	floor.3m	integer no greater than
fopen	fopen.3s	open a stream
fprintf	printf.3s	formatted output conversion
fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream

fread	fread.3s	buffered binary input/output
free	malloc.3	memory allocator
frexp	frexp.3	split into mantissa and exponent
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3s	reposition a stream
ftell	fseek.3s	reposition a stream
ftime	time.3c	get date and time
fwrite	fread.3s	buffered binary input/output
gcvt	ecvt.3	output conversion
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
getdiskbyname	getdisk.3x	get disk description by its name
getenv	getenv.3	value for environment name
getfsent	getfsent.3x	get file system descriptor file entry
getfsfile	getfsent.3x	get file system descriptor file entry
getfsspec	getfsent.3x	get file system descriptor file entry
getfstype	getfsent.3x	get file system descriptor file entry
getgrent	getgrent.3	get group file entry
getgrgid	getgrent.3	get group file entry
getgrnam	getgrent.3	get group file entry
gethostbyaddr	gethostbyname.3n	get network host entry
gethostbyname	gethostbyname.3n	get network host entry
gethostent	gethostbyname.3n	get network host entry
getlogin	getlogin.3	get login name
getnetbyaddr	getnetent.3n	get network entry
getnetbyname	getnetent.3n	get network entry
getnetent	getnetent.3n	get network entry
getpass	getpass.3	read a password
getprotobyname	getprotoent.3n	get protocol entry
getprotobynumber	getprotoent.3n	get protocol entry
getprotoent	getprotoent.3n	get protocol entry
getpw	getpw.3	get name from uid
getpwent	getpwent.3	get password file entry
getpwnam	getpwent.3	get password file entry
getpwuid	getpwent.3	get password file entry
gets	gets.3s	get a string from a stream
getservbyname	getservent.3n	get service entry
getservbyport	getservent.3n	get service entry
getservent	getservent.3n	get service entry
getw	getc.3s	get character or word from stream
getwd	getwd.3	get current working directory pathname
gmtime	ctime.3	convert date and time to ASCII
gtty	stty.3c	set and get terminal state (defunct)
htonl	byteorder.3n	convert values between host and network byte order
htons	byteorder.3n	convert values between host and network byte order
hypot	hypot.3m	Euclidean distance
index	string.3	string operations
inet_addr	inet.3n	Internet address manipulation routines
inet_lnaof	inet.3n	Internet address manipulation routines
inet_makeaddr	inet.3n	Internet address manipulation routines
inet_netof	inet.3n	Internet address manipulation routines
inet_network	inet.3n	Internet address manipulation routines

infnan	infnan.3m	signals exceptions
initgroups	initgroups.3x	initialize group access list
initstate	random.3	better random number generator
insque	insque.3	insert/remove element from a queue
isalnum	ctype.3	character classification macros
isalpha	ctype.3	character classification macros
isascii	ctype.3	character classification macros
isatty	ttyname.3	find name of a terminal
iscntrl	ctype.3	character classification macros
isdigit	ctype.3	character classification macros
islower	ctype.3	character classification macros
isprint	ctype.3	character classification macros
ispunct	ctype.3	character classification macros
isspace	ctype.3	character classification macros
isupper	ctype.3	character classification macros
j0	j0.3m	bessel function
j1	j0.3m	bessel function
jn	j0.3m	bessel function
label	plot.3x	graphics interface
ldexp	frexp.3	split into mantissa and exponent
lgamma	lgamma.3m	log gamma function; (formerly gamma.3m)
lib2648	lib2648.3x	subroutines for the HP 2648 graphics terminal
line	plot.3x	graphics interface
linemod	plot.3x	graphics interface
localtime	ctime.3	convert date and time to ASCII
log	exp.3m	natural logarithm
logb	ieee.3m	exponent extraction
log10	exp.3m	logarithm to base 10
log1p	exp.3m	log(1+x)
longjmp	setjmp.3	non-local goto
malloc	malloc.3	memory allocator
mktemp	mktemp.3	make a unique file name
modf	frexp.3	split into mantissa and exponent
moncontrol	monitor.3	prepare execution profile
monitor	monitor.3	prepare execution profile
monstartup	monitor.3	prepare execution profile
move	plot.3x	graphics interface
nextkey	dbm.3x	data base subroutines
nice	nice.3c	set program priority
nlist	nlist.3	get entries from name list
ntohl	byteorder.3n	convert values between host and network byte order
ntohs	byteorder.3n	convert values between host and network byte order
opendir	directory.3	directory operations
openlog	syslog.3	control system log
openpl	plot.3x	graphics interface
pause	pause.3c	stop until signal
pclose	popen.3	initiate I/O to/from a process
perror	perror.3	system error messages
point	plot.3x	graphics interface
popen	popen.3	initiate I/O to/from a process
pow	exp.3m	exponential x**y
printf	printf.3s	formatted output conversion

psignal	psignal.3	system signal messages
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream
qsort	qsort.3	quicker sort
rand	rand.3c	random number generator
random	random.3	better random number generator
rcmd	rcmd.3x	routines for returning a stream to a remote command
re_comp	regex.3	regular expression handler
re_exec	regex.3	regular expression handler
readdir	directory.3	directory operations
realloc	malloc.3	memory allocator
remque	insque.3	insert/remove element from a queue
rewind	fseek.3s	reposition a stream
rewinddir	directory.3	directory operations
rexec	rexec.3x	return stream to a remote command
rindex	string.3	string operations
rint	floor.3m	round to nearest integer
rresvport	rcmd.3x	routines for returning a stream to a remote command
ruserok	rcmd.3x	routines for returning a stream to a remote command
scalb	ieee.3m	exponent adjustment
scandir	scandir.3	scan a directory
scanf	scanf.3s	formatted input conversion
seekdir	directory.3	directory operations
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setegid	setuid.3	set user and group ID
seteuid	setuid.3	set user and group ID
setfsent	getfsent.3x	get file system descriptor file entry
setgid	setuid.3	set user and group ID
setgrent	getgrent.3	get group file entry
sethostent	gethostbyname.3n	get network host entry
setjmp	setjmp.3	non-local goto
setkey	crypt.3	DES encryption
setlinebuf	setbuf.3s	assign buffering to a stream
setnetent	getnetent.3n	get network entry
setprotoent	getprotoent.3n	get protocol entry
setpwent	getpwent.3	get password file entry
setrgid	setuid.3	set user and group ID
setruid	setuid.3	set user and group ID
setservent	getservent.3n	get service entry
setstate	random.3	better random number generator
setuid	setuid.3	set user and group ID
signal	signal.3	simplified software signal facilities
sin	sin.3m	trigonometric function
sinh	sinh.3m	hyperbolic function
sleep	sleep.3	suspend execution for interval
space	plot.3x	graphics interface
sprintf	printf.3s	formatted output conversion
sqrt	sqrt.3m	square root

srand	rand.3c	random number generator
srandom	random.3	better random number generator
sscanf	scanf.3s	formatted input conversion
stdio	intro.3s	standard buffered input/output package
store	dbm.3x	data base subroutines
strcat	string.3	string operations
strcmp	string.3	string operations
strcpy	string.3	string operations
strlen	string.3	string operations
strncat	string.3	string operations
strncmp	string.3	string operations
strncpy	string.3	string operations
stty	stty.3c	set and get terminal state (defunct)
swab	swab.3	swap bytes
sys_errlist	perror.3	system error messages
sys_nerr	perror.3	system error messages
sys_siglist	psignal.3	system signal messages
syslog	syslog.3	control system log
system	system.3	issue a shell command
tan	sin.3m	trigonometric function
tanh	sinh.3m	hyperbolic function
telldir	directory.3	directory operations
tgetent	termcap.3x	terminal independent operation routines
tgetflag	termcap.3x	terminal independent operation routines
tgetnum	termcap.3x	terminal independent operation routines
tgetstr	termcap.3x	terminal independent operation routines
tgoto	termcap.3x	terminal independent operation routines
time	time.3c	get date and time
times	times.3c	get process times
timezone	ctime.3	convert date and time to ASCII
tputs	termcap.3x	terminal independent operation routines
ttyname	ttyname.3	find name of a terminal
ttyslot	ttyname.3	find name of a terminal
ungetc	ungetc.3s	push character back into input stream
utime	utime.3c	set file times
valloc	valloc.3	aligned memory allocator
varargs	varargs.3	variable argument list
vlimit	vlimit.3c	control maximum system resource consumption
vtimes	vtimes.3c	get information about resource utilization
y0	j0.3m	bessel function
y1	j0.3m	bessel function
yn	j0.3m	bessel function

NAME
     abort - generate a fault

DESCRIPTION
     Abort executes an instruction which is illegal in user mode.
     This causes a signal that normally terminates the process
     with a core dump, which may be used for debugging.

SEE ALSO
     adb(1), sigvec(2), exit(2)

DIAGNOSTICS
     Usually ``Illegal instruction - core dumped'' from the
     shell.

BUGS
     The abort() function does not flush standard I/O buffers.
     Use fflush(3S).

NAME
abs - integer absolute value

SYNOPSIS
```
abs(i)
int i;
```

DESCRIPTION
Abs returns the absolute value of its integer operand.

SEE ALSO
floor(3M) for fabs

BUGS
Applying the abs function to the most negative integer generates a result which is the most negative integer. That is,

```
abs(0x80000000)
```

returns 0x80000000 as a result.

NAME

alarm - schedule signal after specified time

SYNOPSIS

```
alarm(seconds)
unsigned seconds;
```

DESCRIPTION

This interface is made obsolete by setitimer(2).

Alarm causes signal SIGALRM, see sigvec(2), to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

sigpause(2), sigvec(2), signal(3C), sleep(3), ualarm(3), usleep(3)

NAME

asinh, acosh, atanh - inverse hyperbolic functions

SYNOPSIS

```
#include <math.h>

double asinh(x)
double x;

double acosh(x)
double x;

double atanh(x)
double x;
```

DESCRIPTION

These functions compute the designated inverse hyperbolic functions for real arguments.

ERROR (due to Roundoff etc.)

These functions inherit much of their error from log1p described in exp(3M). On a VAX, acosh is accurate to about 3 ulps, asinh and atanh to about 2 ulps. An ulp is one Unit in the Last Place carried.

DIAGNOSTICS

Acosh returns the reserved operand on a VAX if the argument is less than 1.

Atanh returns the reserved operand on a VAX if the argument has absolute value bigger than or equal to 1.

SEE ALSO

math(3M), exp(3M), infnan(3M)

AUTHOR

W. Kahan, Kwok-Choi Ng

NAME

     assert - program verification

SYNOPSIS

```
     #include <assert.h>

     assert(expression)
```

DESCRIPTION

Assert is a macro that indicates expression is expected to be true at this point in the program. It causes an exit(2) with a diagnostic comment on the standard output when expression is false (0). Compiling with the cc(1) option -DNDEBUG effectively deletes assert from the program.

DIAGNOSTICS

`Assertion failed: file f line n.' F is the source file and n the source line number of the assert statement.

NAME
     atof, atoi, atol - convert ASCII to numbers

SYNOPSIS
```
     double atof(nptr)
     char *nptr;

     atoi(nptr)
     char *nptr;

     long atol(nptr)
     char *nptr;
```

DESCRIPTION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional `e' or `E' followed by an optionally signed integer.

Atoi and atol recognize an optional string of spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3S)

BUGS

There are no provisions for overflow.

NAME

bcopy, bcmp, bzero, ffs - bit and byte string operations

SYNOPSIS

```
bcopy(src, dst, length)
char *src, *dst;
int length;

bcmp(b1, b2, length)
char *b1, *b2;
int length;

bzero(b, length)
char *b;
int length;

ffs(i)
long i;
```

DESCRIPTION

The functions bcopy, bcmp, and bzero operate on variable length strings of bytes. They do not check for null bytes as the routines in string(3) do.

Bcopy copies length bytes from string src to the string dst.

Bcmp compares byte string b1 against byte string b2, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be length bytes long.

Bzero places length 0 bytes in the string b1.

Ffs find the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1. A return value of 0 indicates the value passed is zero.

BUGS

The bcopy routine take parameters backwards from strcpy.

NAME
     htonl, htons, ntohl, ntohs - convert values between host and
     network byte order

SYNOPSIS
```
     #include <sys/types.h>
     #include <netinet/in.h>

     netlong = htonl(hostlong);
     u_long netlong, hostlong;

     netshort = htons(hostshort);
     u_short netshort, hostshort;

     hostlong = ntohl(netlong);
     u_long hostlong, netlong;

     hostshort = ntohs(netshort);
     u_short hostshort, netshort;
```

DESCRIPTION
     These routines convert 16 and 32 bit quantities between network byte order and host byte order. On machines such as the SUN these routines are defined as null macros in the include file <netinet/in.h>.

     These routines are most often used in conjunction with Internet addresses and ports as returned by gethostbyname(3N) and getservent(3N).

SEE ALSO
     gethostbyname(3N), getservent(3N)

BUGS
     The VAX handles bytes backwards from most everyone else in the world. This is not expected to be fixed in the near future.

```
NAME
     memccpy, memchr, memcmp, memcpy, memset, strchr, strrchr,
     tempnam, tmpfile, tmpnam - System V compatibility routines

SYNOPSIS
     char *memccpy(from, to, ch, count)
     char *from, *to;
     int ch, count;

     char *memchr(str, ch, count)
     char *str;
     int ch, count;

     int memcmp(str1, str2, count)
     char *str1, *str2;
     int count;

     char *memcpy(from, to, count)
     char *from, to;
     int count;

     char *memset(str, ch, count)
     char *str;
     int ch, count;

     char *strchr(str, ch);
     char *str;
     int ch;

     char *tempnam(tmpdir, prefix)
     char *tmpdir, *prefix;

     char *tmpfile()

     char *tmpnam(str)
     char *str;

COMMENT
     The #defines P_tmpdir and L_tmpnam, used by the routines
     tempnam, tmpfile, and tmpnam are not available in <stdio.h>.
     If the code requires them, just use:

     #include <sys/param.h>
     #define   P_tmpdir  "/usr/tmp"
     #define   L_tmpnam  MAXPATHLEN

     Also, note that the caveat in the System V manual page that
     these functions can start recycling previously used names is
     untrue in this system.

DESCRIPTION
     The above routines are available and behave as in System V.
```

strchr() and strrchr() are simply an alternate entry points into index() and rindex() respectively.

NAME
     creat - create a new file

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

     creat(name, mode)
       char *name;
       int mode;
```

DESCRIPTION
     This interface is made obsolete by: open(2).

     Creat() is the same as:

```
       open(name, O_CREAT | O_TRUNC | O_WRONLY, mode);
```

SEE ALSO
     open(2)

HISTORY
     The creat function call appeared in Version 6 AT&T UNIX.

NAME
     crypt, setkey, encrypt - DES encryption

SYNOPSIS
```
     char *crypt(key, salt)
     char *key, *salt;

     setkey(key)
     char *key;

     encrypt(block, edflag)
     char *block;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to crypt is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The salt string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of setkey is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the encrypt entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by setkey. The edflag flag is ignored; the argument can only be encrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ctime, localtime, gmtime, asctime, timezone, tzset - convert date and time to ASCII

SYNOPSIS

```
void tzset()

char *ctime(clock)
time_t *clock;

#include <time.h>

char *asctime(tm)
struct tm *tm;

struct tm *localtime(clock)
time_t *clock;

struct tm *gmtime(clock)
time_t *clock;

char *timezone(zone, dst)
```

DESCRIPTION

Tzset uses the value of the environment variable TZ to set up the time conversion information used by localtime.

If TZ does not appear in the environment, the TZDEFAULT file (as defined in tzfile.h) is used by localtime. If this file fails for any reason, the GMT offset as provided by the kernel is used. In this case, DST is ignored, resulting in the time being incorrect by some amount if DST is currently in effect. If this fails for any reason, GMT is used.

If TZ appears in the environment but its value is a null string, Greenwich Mean Time is used; if TZ appears and begins with a slash, it is used as the absolute pathname of the tzfile(5)-format file from which to read the time conversion information; if TZ appears and begins with a character other than a slash, it's used as a pathname relative to the system time conversion information directory, defined as TZDIR in the include file tzfile.h. If this file fails for any reason, GMT is used.

Programs that always wish to use local wall clock time should explicitly remove the environmental variable TZ with unsetenv(3).

Ctime converts a long integer, pointed to by clock, such as returned by time(2) into ASCII 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
```

Localtime and gmtime return pointers to structures containing the broken-down time.   Localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time UNIX uses. Asctime converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
   struct tm {
        int tm_sec;    /* 0-59  seconds */
        int tm_min;    /* 0-59  minutes */
        int tm_hour;   /* 0-23  hour */
        int tm_mday;   /* 1-31  day of month */
        int tm_mon;    /* 0-11  month */
        int tm_year;   /* 0-    year - 1900 */
        int tm_wday;   /* 0-6   day of week (Sunday = 0) */
        int tm_yday;   /* 0-365 day of year */
        int tm_isdst;  /* flag: daylight savings time in effect */
        char **tm_zone;        /* abbreviation of timezone name */
        long tm_gmtoff;        /* offset from GMT in seconds */
   };
```

Tm_isdst is non-zero if a time zone adjustment such as Daylight Savings time is in effect.

Tm_gmtoff is the offset (in seconds) of the time represented from GMT, with positive values indicating East of Greenwich.

Timezone remains for compatibility reasons only; it's impossible to reliably map timezone's arguments (zone, a "minutes west of GMT" value and dst, a "daylight saving time in effect" flag) to a time zone abbreviation.

If the environmental string TZNAME exists, timezone returns its value, unless it consists of two comma separated strings, in which case the second string is returned if dst is non-zero, else the first string. If TZNAME doesn't exist, zone is checked for equality with a built-in table of values, in which case timezone returns the time zone or daylight time zone abbreviation associated with that value. If the requested zone does not appear in the table, the difference from GMT is returned; e.g. in Afghanistan, timezone(-(60*4+30), 0) is appropriate because it is 4:30 ahead of GMT, and the string GMT+4:30 is returned.   Programs that in the past used the timezone function should return the zone name as set by localtime to assure correctness.

FILES

/usr/share/zoneinfotime zone information directory

```
     /etc/localtime    local time zone file
```

SEE ALSO
     gettimeofday(2), getenv(3), time(3), tzfile(5), environ(7)

NOTE
     The return values point to static data whose content is overwritten by each call.   The tm_zone field of a returned struct tm points to a static array of characters, which will also be overwritten at the next call (and by calls to tzset).

NAME
     isalpha, isupper, islower, isdigit, isxdigit, isalnum,
     isspace, ispunct, isprint, isgraph, iscntrl, isascii,
     toupper, tolower, toascii - character classification macros

SYNOPSIS
```
     #include <ctype.h>

     isalpha(c)

     . . .
```

DESCRIPTION
     These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii and toascii are defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (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
isxdigit	c is a hex digit
isalnum	c is an alphanumeric character
isspace	c is a space, tab, carriage return, newline, vertical tab, or formfeed
ispunct	c is a punctuation character (neither control nor alphanumeric)
isprint	c is a printing character, code 040(8) (space) through 0176 (tilde)
isgraph	c is a printing character, similar to isprint except false for space.
iscntrl	c is a delete character (0177) or ordinary control character (less than 040).
isascii	c is an ASCII character, code less than 0200
tolower	c is converted to lower case. Return value is undefined if not isupper(c).
toupper	c is converted to upper case. Return value

```
                is undefined if not islower(c).

     toascii          c is converted to be a valid ascii character.
```

SEE ALSO
     ascii(7)

NAME
    curses - screen functions with ``optimal'' cursor motion

SYNOPSIS
    cc [flags] files -lcurses -ltermcap [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() tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine initscr() must be called before any of the other routines that deal with windows and screens are used. The routine endwin() should be called before exiting.

SEE ALSO
    Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold,
    ioctl(2), getenv(3), tty(4), termcap(5)

AUTHOR
    Ken Arnold

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
cbreak()	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
flusok(win,boolf)	set flush-on-refresh flag for win
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) co-ordinates
inch()	get char at current (y,x) co-ordinates
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(lines,cols,begin_y,begin_x) create a new window
     nl()                               set newline mapping
     nocbreak()                         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,lines,cols,begin_y,begin_x) create a subwindow
     touchline(win,y,sx,ex)             mark line y sx through sy as changed
     touchoverlap(win1,win2)            mark overlap of win1 on win2 as changed
     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) co-ordinates 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
```

BUGS

NAME
     daemon - run in the background

SYNOPSIS
```
     int
     daemon(nochdir, noclose)
       int nochdir, noclose;
```

DESCRIPTION
     The daemon function is for programs wishing to detach themselves from the controlling terminal and run in the background as system daemons.

     Unless the argument nochdir is non-zero, daemon changes the current working directory to the root (``/'').

     Unless the argument noclose is non-zero, daemon will redirect standard input, standard output and standard error to ``/dev/null''.

ERRORS
     The function daemon may fail and set errno for any of the errors specified for the library functions fork(2).

SEE ALSO
     fork(2), ioctl(2).

HISTORY
     The daemon function first appeared in 4.4BSD.

NAME

dbminit, fetch, store, delete, firstkey, nextkey - data base subroutines

SYNOPSIS

```
#include <dbm.h>

typedef struct {
   char *dptr;
   int dsize;
} datum;

dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey()

datum nextkey(key)
datum key;
```

DESCRIPTION

Note: the dbm library has been superceded by ndbm(3), and is now implemented using ndbm. These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option -ldbm.

Keys and contents are described by the datum typedef. A datum specifies a string of dsize bytes pointed to by dptr. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has \`.dir' as its suffix. The second file contains all data and has \`.pag' as its suffix.

Before a database can be accessed, it must be opened by dbminit. At the time of this call, the files file.dir and file.pag must exist. (An empty database is created by creating zero-length \`.dir' and \`.pag' files.)

Once open, the data stored under a key is accessed by fetch and data is placed under a key by store. A key (and its associated contents) is deleted by delete. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of firstkey and nextkey. Firstkey will return the first key in the database. With any key nextkey will return the next key in the database. This code will traverse the data base:

```
for (key = firstkey(); key.dptr != NULL; key =
nextkey(key))
```

DIAGNOSTICS

All functions that return an int indicate errors with negative values. A zero return indicates ok. Routines that return a datum indicate errors with a null (0) dptr.

SEE ALSO

ndbm(3)

BUGS

The `.pag' file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. Store will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by firstkey and nextkey depends on a hashing function, not on anything interesting.

NAME

devname - get device name

SYNOPSIS

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

char *
devname(dev, type)
  dev_t dev;
  mode_t type;
```

DESCRIPTION

The devname function returns a pointer to the name of the block or character device in /dev with a device number of dev , and a file type matching the one encoded in type which must be one of S_IFBLK or S_IFCHR. If no device matches the specified values, or no information is available, the string ?? is returned.

SEE ALSO

stat(2), dev_mkdb(8)

HISTORY

The devname function call appeared in 4.4BSD.

NAME

opendir, readdir, telldir, seekdir, rewinddir, closedir - directory operations

SYNOPSIS

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

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;
```

DESCRIPTION

Opendir opens the directory named by filename and associates a directory stream with it. Opendir returns a pointer to be used to identify the directory stream in subsequent operations. The pointer NULL is returned if filename cannot be accessed, or if it cannot malloc(3) enough memory to hold the whole thing.

Readdir returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid seekdir operation.

Telldir returns the current location associated with the named directory stream.

Seekdir sets the position of the next readdir operation on the directory stream. The new position reverts to the one associated with the directory stream when the telldir operation was performed. Values returned by telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the telldir value may be invalidated due to undetected directory compaction. It is safe to use a previous telldir value immediately after a call to opendir and before any calls to readdir.

Rewinddir resets the position of the named directory stream to the beginning of the directory.

Closedir closes the named directory stream and frees the structure associated with the DIR pointer.

Sample code which searchs a directory for entry ``name'' is:

```
   len = strlen(name);
   dirp = opendir(".");
   for (dp = readdir(dirp); dp != NULL; dp =
readdir(dirp))
        if (dp->d_namlen == len && !strcmp(dp->d_name,
name)) {
             closedir(dirp);
             return FOUND;
        }
   closedir(dirp);
   return NOT_FOUND;
```

SEE ALSO

open(2), close(2), read(2), lseek(2), dir(5)

NAME

ecvt, fcvt, gcvt - output conversion

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 the value to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt (negative means 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. The low-order digit is rounded.

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 ndigits.

Gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It attempts to produce ndigit significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

end, etext, edata - last locations in program

SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of etext is the first address above the program text, edata above the initialized data region, and end above the uninitialized data region.

When execution begins, the program break coincides with end, but it is reset by the routines brk(2), malloc(3), standard input/output (stdio(3S)), the profile (-p) option of cc(1), etc. The current value of the program break is reliably returned by `sbrk(0)', see brk(2).

SEE ALSO

brk(2), malloc(3)

NAME

erf, erfc - error functions

SYNOPSIS

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

DESCRIPTION

Erf(x) returns the error function of x; where

erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt.

Erfc(x) returns 1.0-erf(x).

The entry for erfc is provided because of the extreme loss of relative accuracy if erf(x) is called for large x and the result subtracted from 1.   (e.g. for x = 10, 12 places are lost).

SEE ALSO

math(3M)

NAME
     err, verr , errx , verrx , warn , vwarn , warnx , vwarnx -
     formatted error messages

SYNOPSIS
```
     void
     err(eval, fmt, ...)
       int eval;
       char *fmt;

     void
     verr(eval, fmt, args)
       int eval;
       char *fmt;
       va_list args;

     void
     errx(eval, fmt, ...)
       int eval;
       char *fmt;

     void
     verrx(eval, fmt, args)
       int eval
       char *fmt;
       va_list args;

     void
     warn(fmt, ...)
       char *fmt;

     void
     vwarn(fmt, args)
       char *fmt;
       va_list args;

     void
     warnx(fmt, ...)
       char *fmt;

     void
     vwarnx(fmt, args)
       char *fmt;
       va_list args;
```

DESCRIPTION
     The err and warn family of functions display a formatted
     error message on the standard error output. In all cases,
     the last component of the program name, a colon character,
     and a space are output. If the fmt argument is not NULL,
     the formatted error message, a colon character, and a space
     are output. In the case of the err, verr, warn, and vwarn

functions, the error message string affiliated with the current value of the global variable errno is output. In all cases, the output is followed by a newline character.

The err, verr, errx, and verrx functions do not return, but exit with the value of the argument eval.

EXAMPLES

Display the current errno information string and exit:

```
        if ((p = malloc(size)) == NULL)
           err(1, NULL);
        if ((fd = open(file_name, O_RDONLY, 0)) == -1)
           err(1, "%s", file_name);
```

Display an error message and exit:

```
        if (tm.tm_hour < START_TIME)
           errx(1, "too early, wait until %s", start_time_string);
```

Warn of an error:

```
        if ((fd = open(raw_device, O_RDONLY, 0)) == -1)
           warnx("%s: %s: trying the block device",
             raw_device, strerror(errno));
        if ((fd = open(block_device, O_RDONLY, 0)) == -1)
           err(1, "%s", block_device);
```

SEE ALSO

strerror(3)

HISTORY

The err and warn functions first appeared in 4.4BSD.

NAME
     execl, execv, execle, execlp, execvp, exec, execve, exect,
     environ - execute a file

SYNOPSIS
```
     execl(name, arg0, arg1, ..., argn, 0)
     char *name, *arg0, *arg1, ..., *argn;

     execv(name, argv)
     char *name, *argv[];

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

     exect(name, argv, envp)
     char *name, *argv[], *envp[];

     extern char **environ;
```

DESCRIPTION
     These routines provide various interfaces to the execve system call.   Refer to execve(2) for a description of their properties; only brief descriptions are provided here.

     Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

     The name argument is a pointer to the name of the file to be executed.   The pointers arg[0], arg[1] ... address null-terminated strings. Conventionally arg[0] is the name of the file.

     Two interfaces are available. execl is useful when a known file with known arguments is being called; the arguments to execl are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component).   A 0 argument must end the argument list.

     The execv version is useful when the number of arguments is unknown in advance; the arguments to execv are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

     The exect version is used when the executed file is to be manipulated with ptrace(2). The program is forced to single step a single instruction giving the parent an opportunity to manipulate its state. On the VAX-11 this is done by setting the trace bit in the process status longword. Exect is not available on the PDP-11.

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

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

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

Argv is directly usable in another execv because argv[argc] is 0.

Envp is a pointer to an array of strings that constitute the environment of the process. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell sh(1) passes an environment entry for each global shell variable defined when the program is called. See environ(7) for some conventionally used names. The C run-time start-off routine places a copy of envp in the global cell environ, which is used by execv and execl to pass the environment to any subprograms executed by the current program.

Execlp and execvp are called with the same arguments as execl and execv, but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES

/bin/sh shell, invoked if command file found by execlp or execvp

SEE ALSO

execve(2), fork(2), environ(7), csh(1)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see a.out(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is -1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

BUGS

If execvp is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of argv[0] and argv[-1] will be modified before return.

NAME

exit - terminate a process after flushing any pending output

SYNOPSIS

```
exit(status)
int status;
```

DESCRIPTION

Exit terminates a process after calling the Standard I/O library function _cleanup to flush any buffered output. Exit never returns.

SEE ALSO

exit(2), intro(3)

NAME
     exp, expm1, log, log10, log1p, pow - exponential, logarithm,
     power

SYNOPSIS
```
     #include <math.h>

     double exp(x)
     double x;

     double expm1(x)
     double x;

     double log(x)
     double x;

     double log10(x)
     double x;

     double log1p(x)
     double x;

     double pow(x,y)
     double x,y;
```

DESCRIPTION
     Exp returns the exponential function of x.

     Expm1 returns exp(x)-1 accurately even for tiny x.

     Log returns the natural logarithm of x.

     Log10 returns the logarithm of x to base 10.

     Log1p returns log(1+x) accurately even for tiny x.

     Pow(x,y) returns x**y.

ERROR (due to Roundoff etc.)
     exp(x), log(x), expm1(x) and log1p(x) are accurate to within an ulp, and log10(x) to within about 2 ulps; an ulp is one Unit in the Last Place. The error in pow(x,y) is below about 2 ulps when its magnitude is moderate, but increases as pow(x,y) approaches the over/underflow thresholds until almost as many bits could be lost as are occupied by the floating-point format's exponent field; that is 8 bits for VAX D and 11 bits for IEEE 754 Double. No such drastic loss has been exposed by testing; the worst errors observed have been below 20 ulps for VAX D, 300 ulps for IEEE 754 Double. Moderate values of pow are accurate enough that pow(integer,integer) is exact until it is bigger than 2**56 on a VAX, 2**53 for IEEE 754.

DIAGNOSTICS

Exp, expm1 and pow return the reserved operand on a VAX when the correct value would overflow, and they set errno to ERANGE. Pow(x,y) returns the reserved operand on a VAX and sets errno to EDOM when x < 0 and y is not an integer.

On a VAX, errno is set to EDOM and the reserved operand is returned by log unless x > 0, by log1p unless x > -1.

NOTES

The functions exp(x)-1 and log(1+x) are called expm1 and logp1 in BASIC on the Hewlett-Packard HP-71B and APPLE Macintosh, EXP1 and LN1 in Pascal, exp1 and log1 in C on APPLE Macintoshes, where they have been provided to make sure financial calculations of ((1+x)**n-1)/x, namely expm1(n*log1p(x))/x, will be accurate when x is tiny. They also provide accurate inverse hyperbolic functions.

Pow(x,0) returns x**0 = 1 for all x including x = 0, Infinity (not found on a VAX), and NaN (the reserved operand on a VAX). Previous implementations of pow may have defined x**0 to be undefined in some or all of these cases. Here are reasons for returning x**0 = 1 always:

(1) Any program that already tests whether x is zero (or infinite or NaN) before computing x**0 cannot care whether 0**0 = 1 or not. Any program that depends upon 0**0 to be invalid is dubious anyway since that expression's meaning and, if invalid, its consequences vary from one computer system to another.

(2) Some Algebra texts (e.g. Sigler's) define x**0 = 1 for all x, including x = 0. This is compatible with the convention that accepts a[0] as the value of polynomial

```
p(x) = a[0]*x**0 + a[1]*x**1 + a[2]*x**2 +...+ a[n]*x**n
```

at x = 0 rather than reject a[0]*0**0 as invalid.

(3) Analysts will accept 0**0 = 1 despite that x**y can approach anything or nothing as x and y approach 0 independently. The reason for setting 0**0 = 1 anyway is this:

If x(z) and y(z) are any functions analytic (expandable in power series) in z around z = 0, and if there x(0) = y(0) = 0, then x(z)**y(z) -> 1 as z -> 0.

(4) If 0**0 = 1, then infinity**0 = 1/0**0 = 1 too; and then NaN**0 = 1 too because x**0 = 1 for all finite and infinite x, i.e., independently of x.

SEE ALSO

math(3M), infnan(3M)

AUTHOR

Kwok-Choi Ng, W. Kahan

NAME

fclose, fflush - close or flush a stream

SYNOPSIS

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

DESCRIPTION

Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling exit(3).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3S), setbuf(3S)

DIAGNOSTICS

These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

ferror, feof, clearerr, fileno - stream status inquiries

SYNOPSIS

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero. Unless cleared by clearerr, the end-of-file indication lasts until the stream is closed.

Ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clearerr resets the error and end-of-file indicators on the named stream.

Fileno returns the integer file descriptor associated with the stream, see open(2).

Currently all of these functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3S), open(2)

NAME

fabs, floor, ceil, rint - absolute value, floor, ceiling, and round-to-nearest functions

SYNOPSIS

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double x;

double rint(x)
double x;
```

DESCRIPTION

Fabs returns the absolute value |x|.

Floor returns the largest integer no greater than x.

Ceil returns the smallest integer no less than x.

Rint returns the integer (represented as a double precision number) nearest x in the direction of the prevailing rounding mode.

NOTES

On a VAX, rint(x) is equivalent to adding half to the magnitude and then rounding towards zero.

In the default rounding mode, to nearest, on a machine that conforms to IEEE 754, rint(x) is the integer nearest x with the additional stipulation that if |rint(x)-x|=1/2 then rint(x) is even. Other rounding modes can make rint act like floor, or like ceil, or round towards zero.

Another way to obtain an integer near x is to declare (in C)

```
   double x; int k;        k = x;
```

Most C compilers round x towards 0 to get the integer k, but some do otherwise. If in doubt, use floor, ceil, or rint first, whichever you intend. Also note that, if x is larger than k can accommodate, the value of k and the presence or absence of an integer overflow are hard to predict.

SEE ALSO

abs(3), ieee(3M), math(3M)

NAME
     fopen, freopen, fdopen - open a stream

SYNOPSIS
```
     #include <stdio.h>

     FILE *fopen(filename, type)
     char *filename, *type;

     FILE *freopen(filename, type, stream)
     char *filename, *type;
     FILE *stream;

     FILE *fdopen(fildes, type)
     char *type;
```

DESCRIPTION

Fopen opens the file named by filename and associates a stream with it. Fopen returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r" open for reading

"w" create for writing

"a" append: open for writing at end of file, or create for writing

In addition, each type may be followed by a "+" to have the file opened for reading and writing. "r+" positions the stream at the beginning of the file, "w+" creates or truncates it, and "a+" positions it at the end. Both reads and writes may be used on read/write streams, with the limitation that an fseek, rewind, or reading an end-of-file must be used between a read and a write or vice-versa.

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

Freopen is typically used to attach the preopened constant names, stdin, stdout, stderr, to specified files.

Fdopen associates a stream with a file descriptor obtained from open, dup, creat, or pipe(2). The type of the stream must agree with the mode of the open file.

SEE ALSO
     open(2), fclose(3)

DIAGNOSTICS

Fopen and freopen return the pointer NULL if filename cannot be accessed, if too many files are already open, or if other resources needed cannot be allocated.

BUGS

Fdopen is not portable to systems other than UNIX.

The read/write types do not exist on all systems. Those systems without read/write modes will probably treat the type as if the "+" was not present. These are unreliable in any event.

In order to support the same number of open files as does the system, fopen must allocate additional memory for data structures using calloc after 20 files have been opened. This confuses some programs which use their own memory allocators. An undocumented routine, f_prealloc, may be called to force immediate allocation of all internal memory except for buffers.

NAME

fread, fwrite - buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

If stream is stdin and the standard output is line buffered, then any partial output line will be flushed before any call to read(2) to satisfy the fread.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S)

DIAGNOSTICS

Fread and fwrite return 0 upon end of file or error.

NAME
     frexp, ldexp, modf - split into mantissa and exponent

SYNOPSIS
```
     double frexp(value, eptr)
     double value;
     int *eptr;

     double ldexp(value, exp)
     double value;

     double modf(value, iptr)
     double value, *iptr;
```

DESCRIPTION
     Frexp returns the mantissa of a double value as a double quantity, x, of magnitude less than 1 and stores an integer n such that value = x*2n indirectly through eptr.

     Ldexp returns the quantity value*2exp.

     Modf returns the positive fractional part of value and stores the integer part indirectly through iptr.

NAME

fseek, ftell, rewind - reposition a stream

SYNOPSIS

```
#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the stream. The new position is at the signed distance offset bytes from the beginning, the current position, or the end of the file, according as ptrname has the value 0, 1, or 2.

Fseek undoes any effects of ungetc(3S).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named stream. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an offset for fseek.

Rewind(stream) is equivalent to fseek(stream, 0L, 0).

SEE ALSO

lseek(2), fopen(3S)

DIAGNOSTICS

Fseek returns -1 for improper seeks, otherwise zero.

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 returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next int (a 32-bit integer on a VAX-11) from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof and ferror(3S) should be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

clearerr(3S), fopen(3S), putc(3S), gets(3S), scanf(3S), fread(3S), ungetc(3S)

DIAGNOSTICS

These functions return the integer constant EOF at end of file, upon read error, or if an attempt is made to read a file not opened by fopen. The end-of-file condition is remembered, even on a terminal, and all subsequent attempts to read will return EOF until the condition is cleared with clearerr(3S).

BUGS

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, `getc(*f++);' doesn't work sensibly.

NAME
     getdiskbyname - get disk description by its name

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

     struct disklabel *
     getdiskbyname(name)
     char *name;
```

DESCRIPTION
 Getdiskbyname takes a disk name (e.g. rm03) and returns a
 structure describing its geometry information and the stan-
 dard disk partition tables. Information obtained from the
 disktab(5) file has the following form:

```
 /*
  * Copyright (c) 1987, 1988, 1993
  *   The Regents of the University of California. All rights reserved.
  *
  * Redistribution and use in source and binary forms, with or without
  * modification, are permitted provided that the following conditions
  * are met:
  * 1. Redistributions of source code must retain the above copyright
  *    notice, this list of conditions and the following disclaimer.
  * 2. Redistributions in binary form must reproduce the above copyright
  *    notice, this list of conditions and the following disclaimer in the
  *    documentation and/or other materials provided with the distribution.
  * 3. All advertising materials mentioning features or use of this
  *    software
  *    must display the following acknowledgement:
  *   This product includes software developed by the University of
  *   California, Berkeley and its contributors.
  * 4. Neither the name of the University nor the names of its contributors
  *    may be used to endorse or promote products derived from this
  *    software
  *    without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  * PURPOSE
  * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  * SUCH DAMAGE.
  *
  *   @(#)disklabel.h     8.1.1 (2.11BSD) 1995/04/13
  */
```

```
#ifndef   _SYS_DISKLABEL_H_
#define   _SYS_DISKLABEL_H_

/*
 * Disk description table, see disktab(5)
 */
#define   _PATH_DISKTAB  "/etc/disktab"
#define   DISKTAB      "/etc/disktab"     /* deprecated */

/*
 * Each disk has a label which includes information about the hardware
 * disk geometry, filesystem partitions, and drive specific information.
 * The label is in block 0 or 1, possibly offset from the beginning
 * to leave room for a bootstrap, etc.
 */

/* XXX these should be defined per controller (or drive) elsewhere, not here!
*/
#define LABELSECTOR 1        /* sector containing label */
#define LABELOFFSET 0        /* offset of label in sector */

#define DISKMAGIC ((u_long) 0x82564557)   /* The disk magic number */
#define   MAXPARTITIONS  8

/*
 * 2.11BSD's disklabels are different than 4.4BSD for a couple reasons:
 *
 *   1) D space is precious in the 2.11 kernel.  Many of the fields do
 *      not need to be a 'long' (or even a 'short'), a 'short' (or 'char')
 *      is more than adequate.  If anyone ever ports the FFS to a PDP11
 *      changing the label format will be the least of the problems.
 *
 *   2) There is no need to support a bootblock more than 512 bytes long.
 *      The hardware (disk bootroms) only read the first sector, thus the
 *      label is always at sector 1 (the second half of the first filesystem
 *      block).
 *
 * Almost all of the fields have been retained but with reduced sizes.  This
 * is for future expansion and to ease the porting of the various utilities
 * which use the disklabel structure.  The 2.11 kernel uses very little other
 * than the partition tables.  Indeed only the partition tables are resident
 * in the kernel address space, the actual label block is allocated external to
 * the kernel and mapped in as needed.
 */

     struct disklabel {
        u_long    d_magic;         /* the magic number */
        u_char    d_type;          /* drive type */
        u_char    d_subtype;       /* controller/d_type specific */
        char d_typename[16];       /* type name, e.g. "eagle" */
        /*
         * d_packname contains the pack identifier and is returned when
         * the disklabel is read off the disk or in-core copy.
         * d_boot0 is the (optional) name of the primary (block 0) bootstrap
         * as found in /mdec.  This is returned when using
         * getdiskbyname(3) to retrieve the values from /etc/disktab.
         */
     #if defined(KERNEL) || defined(STANDALONE)
        char d_packname[16];            /* pack identifier */
     #else
        union {
             char un_d_packname[16];    /* pack identifier */
             char *un_d_boot0;          /* primary bootstrap name */
        } d_un;
     #define d_packname  d_un.un_d_packname
     #define d_boot0        d_un.un_d_boot0
     #endif    /* ! KERNEL or STANDALONE */
                /* disk geometry: */
        u_short   d_secsize;            /* # of bytes per sector */
        u_short   d_nsectors;           /* # of data sectors per track */
        u_short   d_ntracks;            /* # of tracks per cylinder */
        u_short   d_ncylinders;         /* # of data cylinders per unit */
        u_short   d_secpercyl;          /* # of data sectors per cylinder */
        u_long    d_secperunit;         /* # of data sectors per unit */
        /*
         * Spares (bad sector replacements) below
         * are not counted in d_nsectors or d_secpercyl.
         * Spare sectors are assumed to be physical sectors
         * which occupy space at the end of each track and/or cylinder.
         */
        u_short   d_sparespertrack;     /* # of spare sectors per track */
        u_short   d_sparespercyl;       /* # of spare sectors per cylinder */
        /*
         * Alternate cylinders include maintenance, replacement,
         * configuration description areas, etc.
         */
        u_short   d_acylinders;   /* # of alt. cylinders per unit */

                /* hardware characteristics: */
        /*
         * d_interleave, d_trackskew and d_cylskew describe perturbations
         * in the media format used to compensate for a slow controller.
         * Interleave is physical sector interleave, set up by the formatter
         * or controller when formatting.  When interleaving is in use,
         * logically adjacent sectors are not physically contiguous,
         * but instead are separated by some number of sectors.
         * It is specified as the ratio of physical sectors traversed
         * per logical sector.  Thus an interleave of 1:1 implies contiguous
         * layout, while 2:1 implies that logical sector 0 is separated
         * by one sector from logical sector 1.
         * d_trackskew is the offset of sector 0 on track N
         * relative to sector 0 on track N-1 on the same cylinder.
         * Finally, d_cylskew is the offset of sector 0 on cylinder N
         * relative to sector 0 on cylinder N-1.
         */
        u_short   d_rpm;                 /* rotational speed */
        u_char    d_interleave;          /* hardware sector interleave */
        u_char    d_trackskew;           /* sector 0 skew, per track */
        u_char    d_cylskew;             /* sector 0 skew, per cylinder */
        u_char    d_headswitch;          /* head swith time, usec */
        u_short   d_trkseek;             /* track-to-track seek, msec */
        u_short   d_flags;               /* generic flags */
     #define NDDATA 5
        u_long    d_drivedata[NDDATA];   /* drive-type specific information */
     #define NSPARE 5
        u_long    d_spare[NSPARE];       /* reserved for future use */
        u_long    d_magic2;              /* the magic number (again) */
        u_short   d_checksum;            /* xor of data incl. partitions */

                /* filesystem and partition information: */
        u_short   d_npartitions;         /* number of partitions in following */
        u_short   d_bbsize;              /* size of boot area at sn0, bytes */
        u_short   d_sbsize;              /* max size of fs superblock, bytes */
        struct    partition {            /* the partition table */
             u_long     p_size;          /* number of sectors in partition */
             u_long     p_offset;        /* starting sector */
             u_short    p_fsize;         /* filesystem basic fragment size */
             u_char     p_fstype;        /* filesystem type, see below */
             u_char     p_frag;          /* filesystem fragments per block */
        } d_partitions[MAXPARTITIONS];   /* actually may be more */
     };

     /* d_type values: */
     #define   DTYPE_SMD     1           /* SMD, XSMD; VAX hp/up */
     #define   DTYPE_MSCP    2           /* MSCP */
     #define   DTYPE_DEC     3           /* other DEC (rk, rl) */
     #define   DTYPE_SCSI    4           /* SCSI */
     #define   DTYPE_ESDI    5           /* ESDI interface */
     #define   DTYPE_ST506   6           /* ST506 etc. */
     #define   DTYPE_FLOPPY  7           /* floppy */

     #ifdef DKTYPENAMES
     static char *dktypenames[] = {
        "unknown",
        "SMD",
        "MSCP",
        "old DEC",
        "SCSI",
        "ESDI",
        "ST506",
        "floppy",
        0
     };
     #define DKMAXTYPES  (sizeof(dktypenames) / sizeof(dktypenames[0]) - 1)
     #endif

     /*
```

```
      * Filesystem type and version.
      * Used to interpret other filesystem-specific
      * per-partition information.
      */
     #define   FS_UNUSED    0          /* unused */
     #define   FS_SWAP      1          /* swap */
     #define   FS_V6        2          /* Sixth Edition */
     #define   FS_V7        3          /* Seventh Edition */
     #define   FS_SYSV      4          /* System V */
     /*
      * 2.11BSD uses type 5 filesystems even though block numbers are 4 bytes
      * (rather than the packed 3 byte format) and the directory structure is
      * that of 4.3BSD (long filenames).
     */
     #define   FS_V71K      5        /* V7 with 1K blocks (4.1, 2.9, 2.11) */
     #define   FS_V8        6        /* Eighth Edition, 4K blocks */
     #define   FS_BSDFFS    7        /* 4.2BSD fast file system */
     #define   FS_MSDOS     8        /* MSDOS file system */
     #define   FS_BSDLFS    9        /* 4.4BSD log-structured file system */
     #define   FS_OTHER     10       /* in use, but unknown/unsupported */
     #define   FS_HPFS      11       /* OS/2 high-performance file system */
     #define   FS_ISO9660   12       /* ISO 9660, normally CD-ROM */

     #ifdef    DKTYPENAMES
     static char *fstypenames[] = {
        "unused",
        "swap",
        "Version 6",
        "Version 7",
        "System V",
        "2.11BSD",
        "Eighth Edition",
        "4.2BSD",
        "MSDOS",
        "4.4LFS",
        "unknown",
        "HPFS",
        "ISO9660",
        0
     };
     #define FSMAXTYPES  (sizeof(fstypenames) / sizeof(fstypenames[0]) - 1)
     #endif

     /*
      * flags shared by various drives:
      */
     #define          D_REMOVABLE  0x01       /* removable media */
     #define          D_ECC        0x02       /* supports ECC */
     #define          D_BADSECT    0x04       /* supports bad sector forw. */
     #define          D_RAMDISK    0x08       /* disk emulator */

     /*
```

```
      * Structure used to perform a format
      * or other raw operation, returning data
      * and/or register values.
      * Register identification and format
      * are device- and driver-dependent.
      */
     struct format_op {
        char *df_buf;
        int  df_count;       /* value-result */
        daddr_t   df_startblk;
        int  df_reg[8];      /* result */
     };

     /*
      * Structure used internally to retrieve
      * information about a partition on a disk.
      */
     struct partinfo {
        struct disklabel *disklab;
        struct partition *part;
     };

     /*
      * Disk-specific ioctls.
      */
             /* get and set disklabel; DIOCGPART used internally */
     #define DIOCGDINFO  _IOR(d, 101, struct disklabel)  /* get */
     #define DIOCSDINFO  _IOW(d, 102, struct disklabel)  /* set */
     #define DIOCWDINFO  _IOW(d, 103, struct disklabel)  /* set, update disk */
     #define DIOCGPART   _IOW(d, 104, struct partinfo)   /* get partition */

     /* do format operation, read or write */
     #define DIOCRFORMAT _IOWR(d, 105, struct format_op)
     #define DIOCWFORMAT _IOWR(d, 106, struct format_op)

     #define DIOCSSTEP    _IOW(d, 107, int)   /* set step rate */
     #define DIOCSRETRIES _IOW(d, 108, int)   /* set # of retries */
     #define DIOCWLABEL   _IOW(d, 109, int)   /* write en/disable label */

     #define DIOCSBAD    __IOW(d, 110, struct dkbad)    /* set kernel dkbad */

     #ifndef   KERNEL
     struct disklabel *getdiskbyname();
     #endif

     #if  defined(KERNEL) && !defined(SUPERVISOR)
     memaddr   disklabelalloc();
     #define   LABELDESC (((btoc(sizeof (struct disklabel)) - 1) << 8) | RW)
     #endif

     #endif    /* !_SYS_DISKLABEL_H_ */
```

SEE ALSO
     disktab(5), disklabel(8)

BUGS
     This routine is primarily used when the underlying disk driver does not support labels. The other use is for supplying default information to the disklabel(8) program.

NAME

     getenv, setenv, unsetenv - manipulate environmental vari-
     ables

SYNOPSIS

```
char *getenv(name)
char *name;

setenv(name, value, overwrite)
char *name, *value;
int overwrite;

void unsetenv(name)
char *name;
```

DESCRIPTION

Getenv searches the environment list (see environ(7)) for a string of the form name=value and returns a pointer to the string value if such a string is present, and 0 (NULL) if it is not.

Setenv searches the environment list as getenv does; if the string name is not found, a string of the form name=value is added to the environment.   If it is found, and overwrite is non-zero, its value is changed to value. Setenv returns 0 on success and -1 on failure, where failure is caused by an inability to allocate space for the environment.

Unsetenv removes all occurrences of the string name from the environment. There is no library provision for completely removing the current environment. It is suggested that the following code be used to do so.

```
static char     *envinit[1];
extern char     **environ;
environ = envinit;
```

All of these routines permit, but do not require, a trailing equals (``='') sign on name or a leading equals sign on value.

SEE ALSO

     csh(1), sh(1), execve(2), environ(7)

NAME
     getfsent, getfsspec, getfsfile, getfstype, setfsent,
     endfsent - get file system descriptor file entry

SYNOPSIS
```
     #include <fstab.h>

     struct fstab *getfsent()

     struct fstab *getfsspec(spec)
     char *spec;

     struct fstab *getfsfile(file)
     char *file;

     struct fstab *getfstype(type)
     char *type;

     int setfsent()

     int endfsent()
```

DESCRIPTION
     Getfsent, getfsspec, getfstype, and getfsfile each return a
     pointer to an object with the following structure containing
     the broken-out fields of a line in the file system descrip-
     tion file, <fstab.h>.

```
        struct fstab {
             char *fs_spec;
             char *fs_file;
             char *fs_type;
             int  fs_freq;
             int  fs_passno;
        };
```

     The fields have meanings described in fstab(5).

     Getfsent reads the next line of the file, opening   the file
     if necessary.

     Setfsent opens and rewinds the file.

     Endfsent closes the file.

     Getfsspec and getfsfile sequentially search from the begin-
     ning of the file until a matching special file name or file
     system file name is found, or until EOF is encountered.
     Getfstype   does likewise, matching on the file system type
     field.

FILES
     /etc/fstab

SEE ALSO
     fstab(5)

DIAGNOSTICS
     Null pointer (0) returned on EOF or error.

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

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent - get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

setgrent()

endgrent()
```

DESCRIPTION

Getgrent, getgrgid and getgrnam each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
/*   grp.h      4.1  83/05/03   */

struct    group { /* see getgrent(3) */
     char *gr_name;
     char *gr_passwd;
     int  gr_gid;
     char **gr_mem;
};

struct group *getgrent(), *getgrgid(), *getgrnam();
```

The members of this structure are:

gr_name	The name of the group.
gr_passwd	The encrypted password of the group.
gr_gid	The numerical group-ID.
gr_mem	Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while getgrgid and getgrnam search until a matching gid or name is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to setgrent has the effect of rewinding the group file to allow repeated searches. Endgrent may be called to close the group file when processing is complete.

FILES
     /etc/group

SEE ALSO
     getlogin(3), getpwent(3), group(5)

DIAGNOSTICS
     A null pointer (0) is returned on EOF or error.

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

NAME

getgrouplist - calculate group access list

SYNOPSIS

```
#include <unistd.h>
int
getgrouplist(name, basegid, groups, ngroups)
        char     *name;
        gid_t basegid;
        gid_t *groups;
        int *ngroups;
```

DESCRIPTION

The getgrouplist function reads through the group file and calculates the group access list for the user specified in name. The basegid is automatically included in the groups list. Typically this value is given as the group number from the password file.

The resulting group list is returned in the integer array pointed to by groups. The caller specifies the size of the groups array in the integer pointed to by ngroups; the actual number of groups found is returned in ngroups.

RETURN VALUES

The getgrouplist function returns -1 if the size of the group list is too small to hold all the user's groups. Here, the group array will be filled with as many groups as will fit.

FILES

/etc/group group membership list

SEE ALSO

setgroups(2), initgroups(3)

HISTORY

The getgrouplist function first appeared in 4.4BSD.

BUGS

The getgrouplist function uses the routines based on getgrent(3). If the invoking program uses any of these routines, the group structure will be overwritten in the call to getgrouplist.

NAME
     gethostbyname, gethostbyaddr, gethostent, sethostent,
     endhostent, herror - get network host entry

SYNOPSIS
```
     #include <netdb.h>

     extern int h_errno;

     struct hostent *gethostbyname(name)
     char *name;

     struct hostent *gethostbyaddr(addr, len, type)
     char *addr; int len, type;

     struct hostent *gethostent()

     sethostent(stayopen)
     int stayopen;

     endhostent()

     herror(string)
     char *string;
```

DESCRIPTION
     Gethostbyname and gethostbyaddr each return a pointer to an
     object with the following structure describing an internet
     host referenced by name or by address, respectively. This
     structure contains either the information obtained from the
     name server, named(8), or broken-out fields from a line in
     /etc/hosts. If the local name server is not running these
     routines do a lookup in /etc/hosts.

```
        struct    hostent {
             char *h_name;         /* official name of host */
             char **h_aliases;     /* alias list */
             int  h_addrtype;      /* host address type */
             int  h_length;        /* length of address */
             char **h_addr_list;   /* list of addresses from name server */
        };
        #define   h_addr  h_addr_list[0]/* address,for backward compatibility */
```

     The members of this structure are:

     h_name   Official name of the host.

     h_aliases   A zero terminated array of alternate names for
             the host.

     h_addrtype   The type of address being returned;   currently
             always AF_INET.

h_length   The length, in bytes, of the address.

h_addr_list A zero terminated array of network addresses for the host. Host addresses are returned in network byte order.

h_addr The first address in h_addr_list; this is for backward compatiblity.

When using the nameserver, gethostbyname will search for the named host in the current domain and its parents unless the name ends in a dot. If the name contains no dot, and if the environment variable ``HOSTALIASES'' contains the name of an alias file, the alias file will first be searched for an alias matching the input name. See hostname(7) for the domain search procedure and the alias file format.

Sethostent may be used to request the use of a connected TCP socket for queries. If the stayopen flag is non-zero, this sets the option to send all queries to the name server using TCP and to retain the connection after each call to gethost-byname or gethostbyaddr. Otherwise, queries are performed using UDP datagrams.

Endhostent closes the TCP connection.

DIAGNOSTICS

Error return status from gethostbyname and gethostbyaddr is indicated by return of a null pointer. The external integer h_errno may then be checked to see whether this is a temporary failure or an invalid or unknown host. The routine herror can be used to print an error message describing the failure. If its argument string is non-NULL, it is printed, followed by a colon and a space. The error message is printed with a trailing newline.

h_errno can have the following values:

HOST_NOT_FOUND No such host is known.

TRY_AGAIN This is usually a temporary error and means that the local server did not receive a response from an authoritative server. A retry at some later time may succeed.

NO_RECOVERY Some unexpected server failure was encountered. This is a non-recoverable error.

NO_DATA The requested name is valid but does not have an IP address; this is not a temporary error. This means that the name is known to the name server but there is no address associated with this name. Another type of request to the name server using this domain name will result in an answer; for example, a mail-forwarder may be registered for this domain.

FILES

/etc/hosts

SEE ALSO

resolver(3), hosts(5), hostname(7), named(8)

CAVEAT

Gethostent is defined, and sethostent and endhostent are redefined, when libc is built to use only the routines to lookup in /etc/hosts and not the name server.

Gethostent reads the next line of /etc/hosts, opening the file if necessary.

Sethostent is redefined to open and rewind the file. If the stayopen argument is non-zero, the hosts data base will not be closed after each call to gethostbyname or gethostbyaddr. Endhostent is redefined to close the file.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

getloadavg - get system load averages

SYNOPSIS

```
     getloadavg(loadavg, nelem)
          double loadavg[];
          int    nelem;
```

DESCRIPTION

The getloadavg function returns the number of processes in the system run queue averaged over various periods of time. Up to nelem samples are retrieved and assigned to successive elements of loadavg. The system imposes a maximum of 3 samples, representing averages over the last 1, 5, and 15 minutes, respectively.

DIAGNOSTICS

If the load average was unobtainable, -1 is returned; otherwise, the number of samples actually retrieved is returned.

SEE ALSO

uptime(1), sysctl(3), sysctl(8).

HISTORY

The getloadavg function appeared in 4.3BSD-Reno.

This function replaces the 2.9BSD compatibility routine getla().

NAME
     getmntinfo - get information about mounted file systems

SYNOPSIS
```
     #include <sys/param.h>
     #include <sys/mount.h>

     int
     getmntinfo(mntbufp,flags)
     struct statfs **mntbufp;
     int flags
```

DESCRIPTION
     The getmntinfo() function returns an array of statfs structures describing each currently mounted file system (see statfs(2).

     The getmntinfo() function passes its flags parameter transparently to getfsstat(2).

RETURN VALUES
     On successful completion, getmntinfo() returns a count of the number of elements in the array. The pointer to the array is stored into mntbufp.

     If an error occurs, zero is returned and the external variable errno is set to indicate the error. Although the pointer mntbufp will be unmodified, any information previously returned by getmntinfo() will be lost.

ERRORS
     The getmntinfo() function may fail and set errno for any of the errors specified for the library routines getfsstat(2) or malloc(3).

SEE ALSO
     getfsstat(2), statfs(2), mount(2), mount(8)

HISTORY
     The getmntinfo() function first appeared in 4.4BSD.

BUGS
     The getmntinfo() function writes the array of structures to an internal static object and returns a pointer to that object. Subsequent calls to getmntinfo() will modify the same object.

     The memory allocated by getmntinfo() cannot be free(2)'d by the application.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent - get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent()

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net, type)
long net;
int type;

setnetent(stayopen)
int stayopen;

endnetent()
```

DESCRIPTION

Getnetent, getnetbyname, and getnetbyaddr each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, /etc/networks.

```
struct    netent {
     char *n_name;             /* official name of net */
     char **n_aliases;         /* alias list */
     int   n_addrtype;         /* net number type */
     unsigned long  n_net;     /* net number */
};
```

The members of this structure are:

n_name The official name of the network.

n_aliases   A zero terminated list of alternate names for the network.

n_addrtype The type of the network number returned; currently only AF_INET.

n_net The network number. Network numbers are returned in machine byte order.

Getnetent reads the next line of the file, opening the file if necessary.

Setnetent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getnetbyname or getnetbyaddr.

Endnetent closes the file.

Getnetbyname and getnetbyaddr sequentially search from the beginning of the file until a matching net name or net address and type is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NAME

getopt - get option character from command line argument list

SYNOPSIS

```
     #include <stdlib.h>

     extern char *optarg;
     extern int optind;
     extern int optopt;
     extern int opterr;
     extern int optreset;

     int
     getopt(argc, argv, optstring)
     int argc;
     char **argv;
     char *optstring;
```

DESCRIPTION

The getopt() function incrementally parses a command line argument list argv and returns the next known option character. An option character is known if it has been specified in the string of accepted option characters, optstring.

The option string optstring may contain the following elements: individual characters, and characters followed by a colon to indicate an option argument is to follow. For example, an option string """x"" recognizes an option ``-x'', and an option string """x:"" recognizes an option and argument ``-x argument''. It does not matter to getopt() if a following argument has leading white space.

On return from getopt(), optarg points to an option argument, if it is anticipated, and the variable optind contains the index to the next argv argument for a subsequent call to getopt().   The variable optopt saves the last known option character returned by getopt().

The variable opterr and optind are both initialized to 1. The optind variable may be set to another value before a set of calls to getopt() in order to skip over more or less argv entries.

In order to use getopt() to evaluate multiple sets of arguments, or to evaluate a single set of arguments multiple times, the variable optreset must be set to 1 before the second and each additional set of calls to getopt(), and the variable optind must be reinitialized.

The getopt() function returns an EOF when the argument list is exhausted, or a non-recognized option is encountered.

The interpretation of options in the argument list may be cancelled by the option `--' (double dash) which causes getopt() to signal the end of argument processing and return an EOF. When all options have been processed (i.e., up to the first non-option argument), getopt() returns EOF.

DIAGNOSTICS

If the getopt() function encounters a character not found in the string optarg or detects a missing option argument it writes an error message and returns `?' to the stderr. Setting opterr to a zero will disable these error messages. If optstring has a leading `:' then a missing option argument causes a `:' to be returned in addition to suppressing any error messages.

Option arguments are allowed to begin with `-'; this is reasonable but reduces the amount of error checking possible.

EXTENSIONS

The optreset variable was added to make it possible to call the getopt() function multiple times. This is an extension to the IEEE Std1003.2 (``POSIX'') specification.

EXAMPLE

```
extern char *optarg;
extern int optind;
int bflag, ch, fd;

bflag = 0;
while ((ch = getopt(argc, argv, "bf:")) != EOF)
   switch(ch) {
   case 'b':
        bflag = 1;
        break;
   case 'f':
        if ((fd = open(optarg, O_RDONLY, 0)) < 0) {
           (void)fprintf(stderr,
             "myname: %s: %s\n", optarg, strerror(errno));
           exit(1);
        }
        break;
   case '?':
   default:
        usage();
}
argc -= optind;
argv += optind;
```

HISTORY

The getopt() function appeared 4.3BSD.

BUGS

A single dash ``-'' may be specified as an character in opt- string , however it should never have an argument associated with it. This allows getopt() to be used with programs that expect ``-'' as an option flag. This practice is wrong, and should not be used in any current development. It is provided for backward compatibility only . By default, a single dash causes getopt() to return EOF. This is, we believe, compatible with System V.

It is also possible to handle digits as option letters. This allows getopt() to be used with programs that expect a number (``-3'') as an option. This practice is wrong, and should not be used in any current development. It is provided for backward compatibility only. The following code fragment works in most cases.

```
int length;
char *p;

while ((c = getopt(argc, argv, "0123456789")) != EOF)
   switch (c) {
   case '0': case '1': case '2': case '3': case '4':
   case '5': case '6': case '7': case '8': case '9':
        p = argv[optind - 1];
        if (p[0] == '-' && p[1] == ch && !p[2])
           length = atoi(++p);
        else
           length = atoi(argv[optind] + 1);
        break;
   }
}
```

NAME
     getpass - read a password

SYNOPSIS
```
     char *getpass(prompt)
     char *prompt;
```

DESCRIPTION
     Getpass reads a password from the file /dev/tty, or if that cannot be opened, from the standard input, after prompting with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES
     /dev/tty

SEE ALSO
     crypt(3)

BUGS
     The return value points to static data whose content is overwritten by each call.

```
NAME
     getprotoent, getprotobynumber, getprotobyname, setprotoent,
     endprotoent - get protocol entry

SYNOPSIS
     #include <netdb.h>

     struct protoent *getprotoent()

     struct protoent *getprotobyname(name)
     char *name;

     struct protoent *getprotobynumber(proto)
     int proto;

     setprotoent(stayopen)
     int stayopen

     endprotoent()

DESCRIPTION
     Getprotoent, getprotobyname, and getprotobynumber each
     return a pointer to an object with the following structure
     containing the broken-out fields of a line in the network
     protocol data base, /etc/protocols.

        struct    protoent {
             char *p_name;  /* official name of protocol */
             char **p_aliases;   /* alias list */
             int  p_proto;  /* protocol number */
        };

     The members of this structure are:

     p_name The official name of the protocol.

     p_aliases   A zero terminated list of alternate names for the
           protocol.

     p_proto     The protocol number.

     Getprotoent reads the next line of  the  file,  opening  the
     file if necessary.

     Setprotoent opens and rewinds the  file. If  the  stayopen
     flag is non-zero, the net data base will not be closed after
     each call to getprotobyname or getprotobynumber.

     Endprotoent closes the file.

     Getprotobyname and getprotobynumber sequentially search from
     the  beginning of the file until a matching protocol name or
```

protocol number is found, or until EOF is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getpwent, getpwnam, getpwuid, setpassent, setpwfile,
setpwent, endpwent - get password file entries

SYNOPSIS

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

struct passwd *getpwent()

struct passwd *getpwnam(login)
char *login;

struct passwd *getpwuid(uid)
uid_t uid;

int setpassent(stayopen)
int stayopen;

void setpwfile(file)
char *file;

int setpwent()

void endpwent()
```

DESCRIPTION

Getpwent, getpwuid, and getpwnam each return a pointer to a structure containing the broken-out fields of a line in the password file. This structure is defined by the include file <pwd.h>, and contains the following fields:

```
   struct passwd {
        char  *pw_name;       /* user name */
        char  *pw_passwd;     /* encrypted password */
        uid_t  pw_uid;        /* user uid */
        gid_t  pw_gid;        /* user gid */
        time_t pw_change;     /* password change time */
        char  *pw_class;      /* user access class */
        char  *pw_gecos;      /* Honeywell login info */
        char  *pw_dir;        /* home directory */
        char  *pw_shell;      /* default shell */
        time_t pw_expire;     /* account expiration */
   };
```

These fields are more completely described in passwd(5).

Getpwnam and getpwuid search the password database for a matching user name or user uid, respectively, returning the first one encountered. Identical user names or user uids may result in undefined behavior.

Getpwent sequentially reads the password database and is intended for programs that wish to step through the complete list of users.

All three routines will open the password file for reading, if necessary.

Setpwfile changes the default password file to file, thus allowing the use of alternate password files.

Setpassent opens the file or rewinds it if it is already open. If stayopen is non-zero, file descriptors are left open, significantly speeding up subsequent calls.   This functionality is unnecessary for getpwent as it doesn't close its file descriptors by default. It should also be noted that it is dangerous for long-running programs to use this functionality as the password file may be updated by chpass(1), passwd(1), or vipw(8).

Setpwent is identical to setpassent with an argument of zero.

Endpwent closes any open files.

These routines have been written to ``shadow'' the password file, e.g. allow only certain programs to have access to the encrypted password. This is done by using the mkpasswd(8) program, which creates ndbm(3) databases that correspond to the password file, with the single exception that, rather than storing the encrypted password in the database, it stores the offset in the password file where the encrypted password may be found. Getpwent, getpwnam, and getpwuid will use the ndbm files in preference to the ``real'' password files, only reading the password file itself, to obtain the encrypted password, if the process is running with an effective user id equivalent to super-user. If the password file itself is protected, and the ndbm files are not, this makes the password available only to programs running with super-user privileges.

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), ndbm(3), passwd(5)

DIAGNOSTICS

The routines getpwent, getpwnam, and getpwuid, return a null pointer on EOF or error. Setpassent and setpwent return 0 on failure and 1 on success. Endpwent and setpwfile have no return value.

BUGS

All information is contained in a static buffer which is overwritten by each new call. It must be copied elsewhere to be retained.

Intermixing calls to getpwent with calls to getpwnam or getpwuid, or intermixing calls to getpwnam and getpwuid, after using setpassent to require that file descriptors be left open, may result in undefined behavior.

The routines getpwent, endpwent, setpassent, and setpwent are fairly useless in a networked environment and should be avoided, if possible.

NAME

gets, fgets - get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Gets reads a string into s from the standard input stream stdin. The string is terminated by a newline character, which is replaced in s by a null character. Gets returns its argument.

Fgets reads n-1 characters, or up through a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

SEE ALSO

puts(3S), getc(3S), scanf(3S), fread(3S), ferror(3S)

DIAGNOSTICS

Gets and fgets return the constant pointer NULL upon end of file or error.

BUGS

Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.

NAME

getservent, getservbyport, getservbyname, setservent, endservent - get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen

endservent()
```

DESCRIPTION

Getservent, getservbyname, and getservbyport each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, /etc/services.

```
   struct    servent {
        char *s_name;         /* official name of service */
        char **s_aliases;     /* alias list */
        int  s_port;          /* port service resides at */
        char *s_proto;        /* protocol to use */
   };
```

The members of this structure are:

s_name The official name of the service.

s_aliases A zero terminated list of alternate names for the service.

s_port The port number at which the service resides. Port numbers are returned in network byte order.

s_proto The name of the protocol to use when contacting the service.

Getservent reads the next line of the file, opening the file if necessary.

Setservent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getservbyname or .IR getservbyport .

Endservent closes the file.

Getservbyname and getservbyport sequentially search from the beginning of the file until a matching protocol name or port number is found, or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3N), services(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

getsubopt - get sub options from an argument

SYNOPSIS

```
#include <stdlib.h>

extern char *suboptarg

int
getsubopt(optionp, tokens, valuep)
char **optionp;
char **tokens;
char **valuep;
```

DESCRIPTION

The getsubopt() function parses a string containing tokens delimited by one or more tab, space or comma (`,') characters. It is intended for use in parsing groups of option arguments provided as part of a utility command line.

The argument optionp is a pointer to a pointer to the string. The argument tokens is a pointer to a NULL-terminated array of pointers to strings.

The getsubopt() function returns the zero-based offset of the pointer in the tokens array referencing a string which matches the first token in the string, or, -1 if the string contains no tokens or tokens does not contain a matching string.

If the token is of the form ``name=value'', the location referenced by valuep will be set to point to the start of the ``value'' portion of the token.

On return from getsubopt(), optionp will be set to point to the start of the next token in the string, or the null at the end of the string if no more tokens are present. The external variable suboptarg will be set to point to the start of the current token, or NULL if no tokens were present. The argument valuep will be set to point to the ``value'' portion of the token, or NULL if no ``value'' portion was present.

EXAMPLE

```
char *tokens[] = {
   #define   ONE  0
        "one",
   #define   TWO  1
        "two",
   NULL
};
```

```
     ...

     extern char *optarg, *suboptarg;
     char *options, *value;

     while ((ch = getopt(argc, argv, "ab:")) != -1) {
        switch(ch) {
        case 'a':
             /* process ``a'' option */
             break;
        case 'b':
             options = optarg;
             while (*options) {
                switch(getsubopt(&options, tokens, &value)) {
                case ONE:
                   /* process ``one'' sub option */
                   break;
                case TWO:
                   /* process ``two'' sub option */
                   if (!value)
                        error("no value for two");
                   i = atoi(value);
                   break;
                case -1:
                   if (suboptarg)
                        error("illegal sub option %s",
                        suboptarg);
                   else
                        error("missing sub option");
                   break;
             }
             break;
        }
```

SEE ALSO
     getopt(3), strsep(3)

HISTORY
     The getsubopt() function first appeared in 4.4BSD.

NAME
getttyent, getttynam, setttyent, endttyent - get ttys file
entry

SYNOPSIS
```
#include <ttyent.h>

struct ttyent *getttyent()

struct ttyent *getttynam(name)
char *name;

setttyent()

endttyent()
```

DESCRIPTION
Getttyent, and getttynam each return a pointer to an object
with the following structure containing the broken-out
fields of a line from the tty description file.

```
   /*
    * Copyright (c) 1983 Regents of the University of California.
    * All rights reserved.  The Berkeley software License Agreement
    * specifies the terms and conditions for redistribution.
    *
    *   @(#)ttyent.h   5.1 (Berkeley) 5/30/85
    */

   struct    ttyent {         /* see getttyent(3) */
        char *ty_name;        /* terminal device name */
        char *ty_getty;       /* command to execute, usually getty */
        char *ty_type;        /* terminal type for termcap (3X) */
        int  ty_status;       /* status flags (see below for defines) */
        char  *ty_window;     /* command to start up window manager */
        char *ty_comment;     /* usually the location of the terminal */
   };

   #define TTY_ON      0x1  /* enable logins (startup getty) */
   #define TTY_SECURE  0x2  /* allow root to login */

   extern struct ttyent *getttyent();
   extern struct ttyent *getttynam();
```

ty_name   is the name of the character-special file in
the directory ``/dev". For various reasons, it
must reside in the directory ``/dev".

ty_getty   is the command (usually getty(8)) which is
invoked by init to initialize tty line charac-
teristics. In fact, any arbitrary command can
be used; a   typical use is to initiate   a

```
          terminal emulator in a window system.

     ty_type       is the name of the default terminal  type  con-
          nected  to  this  tty line. This is typically a
          name     from  the  termcap(5)  data  base.   The
          environment variable `TERM' is initialized with
          this name by getty(8) or login(1).

     ty_status     is a mask of bit fields which indicate  various
          actions  to  be  allowed  on this tty line. The
          following is a description of each flag.

          TTY_ON       Enables logins (i.e., init(8) will
                    start  the specified ``getty" com-
                    mand on this entry).

          TTY_SECURE   Allows root to login on this  ter-
                    minal.  Note that `TTY_ON' must be
                    included for this to be useful.

     ty_window     is the command to execute for a  window  system
          associated  with  the  line.    The window system
          will be started before the command specified in
          the  ty_getty  entry is  executed.  If none is
          specified, this will be null.

     ty_comment   is the trailing comment field, if any; a  lead-
          ing delimiter and white space will be removed.

     Getttyent reads the next line from the  ttys  file,  opening
     the file if necessary; setttyent rewinds the file; endttyent
     closes it.

     Getttynam searches from the beginning of the  file  until a
     matching name is found (or until EOF is encountered).

FILES
     /etc/ttys

SEE ALSO
     login(1),   ttyslot(3),  ttys(5),  gettytab(5),   termcap(5),
     getty(8), init(8)

DIAGNOSTICS
     Null pointer (0) returned on EOF or error.

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

NAME

getusershell, setusershell, endusershell - get legal user shells

SYNOPSIS

```
char *getusershell()

setusershell()

endusershell()
```

DESCRIPTION

Getusershell returns a pointer to a legal user shell as defined by the system manager in the file /etc/shells. If /etc/shells is unreadable or does not exist, getusershell behaves as if /bin/sh and /bin/csh were listed in the file.

Getusershell reads the next line (opening the file if necessary); setusershell rewinds the file; endusershell closes it.

FILES

/etc/shells

SEE ALSO

shells(5)

DIAGNOSTICS

The routine getusershell returns a null pointer (0) on EOF.

BUGS

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

NAME
     getwd - get current working directory pathname

SYNOPSIS
```
     char *getwd(pathname)
     char *pathname;
```

DESCRIPTION
     Getwd copies the absolute pathname of the current working directory to pathname and returns a pointer to the result.

LIMITATIONS
     Maximum pathname length is MAXPATHLEN characters (1024), as defined in <sys/param.h>.

DIAGNOSTICS
     Getwd returns zero and places a message in pathname if an error occurs.

NAME
hypot, cabs - Euclidean distance, complex absolute value

SYNOPSIS
```
     #include <math.h>

     double hypot(x,y)
     double x,y;

     double cabs(z)
     struct {double x,y;} z;
```

DESCRIPTION
Hypot(x,y) and cabs(x,y) return sqrt(x*x+y*y) computed in such a way that underflow will not happen, and overflow occurs only if the final result deserves it.

hypot(infinity,v) = hypot(v,infinity) = +infinity for all v, including NaN.

ERROR (due to Roundoff, etc.)
Below 0.97 ulps. Consequently hypot(5.0,12.0) = 13.0 exactly; in general, hypot and cabs return an integer whenever an integer might be expected.

The same cannot be said for the shorter and faster version of hypot and cabs that is provided in the comments in cabs.c; its error can exceed 1.2 ulps.

NOTES
As might be expected, hypot(v,NaN) and hypot(NaN,v) are NaN for all finite v; with "reserved operand" in place of "NaN", the same is true on a VAX. But programmers on machines other than a VAX (it has no infinity) might be surprised at first to discover that hypot(+infinity,NaN) = +infinity. This is intentional; it happens because hypot(infinity,v) = +infinity for all v, finite or infinite. Hence hypot(infinity,v) is independent of v. Unlike the reserved operand on a VAX, the IEEE NaN is designed to disappear when it turns out to be irrelevant, as it does in hypot(infinity,NaN).

SEE ALSO
math(3M), sqrt(3M)

AUTHOR
W. Kahan

NAME

copysign, drem, finite, logb, scalb - copysign, remainder, exponent manipulations

SYNOPSIS

```
#include <math.h>

double copysign(x,y)
double x,y;

double drem(x,y)
double x,y;

int finite(x)
double x;

double logb(x)
double x;

double scalb(x,n)
double x;
int n;
```

DESCRIPTION

These functions are required for, or recommended by the IEEE standard 754 for floating-point arithmetic.

Copysign(x,y) returns x with its sign changed to y's.

Drem(x,y) returns the remainder r := x - n*y where n is the integer nearest the exact value of x/y; moreover if |n-x/y|=1/2 then n is even. Consequently the remainder is computed exactly and |r| < |y|/2. But drem(x,0) is exceptional; see below under DIAGNOSTICS.

```
Finite(x) = 1 just when -infinity < x < +infinity,
          = 0 otherwise (when |x| = infinity or x is NaN or
                     x is the VAX's reserved operand.)
```

Logb(x) returns x's exponent n, a signed integer converted to double-precision floating-point and so chosen that 1 < |x|/2**n < 2 unless x = 0 or (only on machines that conform to IEEE 754) |x| = infinity or x lies between 0 and the Underflow Threshold; see below under "BUGS".

Scalb(x,n) = x*(2**n) computed, for integer n, without first computing 2**n.

DIAGNOSTICS

IEEE 754 defines drem(x,0) and drem(infinity,y) to be invalid operations that produce a NaN. On a VAX, drem(x,0) returns the reserved operand. No infinity exists on a VAX.

IEEE 754 defines logb(+infinity) = +infinity and logb(0) = -infinity, requires the latter to signal Division-by-Zero. But on a VAX, logb(0) = 1.0 - 2.0**31 = -2,147,483,647.0. And if the correct value of scalb(x,n) would overflow on a VAX, it returns the reserved operand and sets errno to ERANGE.

SEE ALSO

floor(3M), math(3M), infnan(3M)

AUTHOR

Kwok-Choi Ng

BUGS

Should drem(x,0) and logb(0) on a VAX signal invalidity by setting errno = EDOM? Should logb(0) return -1.7e38?

IEEE 754 currently specifies that logb(denormalized no.) = logb(tiniest normalized no. > 0) but the consensus has changed to the specification in the new proposed IEEE standard p854, namely that logb(x) satisfy

```
  1 < scalb(|x|,-logb(x)) < Radix   ... = 2 for IEEE 754
```

for every x except 0, infinity and NaN. Almost every program that assumes 754's specification will work correctly if logb follows 854's specification instead.

IEEE 754 requires copysign(x,NaN) = +x but says nothing else about the sign of a NaN. A NaN (Not a Number) is similar in spirit to the VAX's reserved operand, but very different in important details. Since the sign bit of a reserved operand makes it look negative,

```
  copysign(x,reserved operand) = -x;
```

should this return the reserved operand instead?

NAME

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof - Internet address manipulation routines

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(cp)
char *cp;

unsigned long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, lna)
long net, lna;

long inet_lnaof(in)
struct in_addr in;

long inet_netof(in)
struct in_addr in;
```

DESCRIPTION

The routines inet_addr and inet_network each interpret character strings representing numbers expressed in the Internet standard "." notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine inet_ntoa takes an Internet address and returns an ASCII string representing the address in "." notation. The routine inet_makeaddr takes an Internet network number and a local network address and constructs an Internet address from it. The routines inet_netof and inet_lnaof break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Values specified using the "." notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
```

a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the VAX the bytes referred to above appear as "d.c.b.a". That is, VAX bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a "." notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e., a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

gethostbyname(3N), getnetent(3N), hosts(5), networks(5),

DIAGNOSTICS

The value -1 is returned by inet_addr and inet_network for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed. The string returned by inet_ntoa resides in a static memory area.
Inet_addr should return a struct in_addr.

NAME

infnan - signals invalid floating-point operations on a VAX (temporary)

SYNOPSIS

```
#include <math.h>

double infnan(iarg)
int iarg;
```

DESCRIPTION

At some time in the future, some of the useful properties of the Infinities and NaNs in the IEEE standard 754 for Binary Floating-Point Arithmetic will be simulated in UNIX on the DEC VAX by using its Reserved Operands. Meanwhile, the Invalid, Overflow and Divide-by-Zero exceptions of the IEEE standard are being approximated on a VAX by calls to a procedure infnan in appropriate places in libm. When better exception-handling is implemented in UNIX, only infnan among the codes in libm will have to be changed. And users of libm can design their own infnan now to insulate themselves from future changes.

Whenever an elementary function code in libm has to simulate one of the aforementioned IEEE exceptions, it calls infnan(iarg) with an appropriate value of iarg. Then a reserved operand fault stops computation. But infnan could be replaced by a function with the same name that returns some plausible value, assigns an apt value to the global variable errno, and allows computation to resume. Alternatively, the Reserved Operand Fault Handler could be changed to respond by returning that plausible value, etc. instead of aborting.

In the table below, the first two columns show various exceptions signaled by the IEEE standard, and the default result it prescribes. The third column shows what value is given to iarg by functions in libm when they invoke infnan(iarg) under analogous circumstances on a VAX. Currently infnan stops computation under all those circumstances. The last two columns offer an alternative; they suggest a setting for errno and a value for a revised infnan to return. And a C program to implement that suggestion follows.

IEEE Signal	IEEE Default	iarg	errno	infnan
Invalid	NaN	EDOM	EDOM	0
Overflow	+Infinity	ERANGE	ERANGE	HUGE
Div-by-0	+Infinity	+ERANGE	ERANGE or EDOM	+HUGE

(HUGE = 1.7e38 ... nearly 2.0**127)

```
     ALTERNATIVE  infnan:
     #include    <math.h>
     #include    <errno.h>
     extern int errno ;
     double infnan(iarg)
     int    iarg ;
     {
            switch(iarg) {
            case    ERANGE: errno = ERANGE; return(HUGE);
            case   -ERANGE: errno = EDOM;   return(-HUGE);
            default:      errno = EDOM;   return(0);
            }
     }
```

SEE ALSO

math(3M), intro(2), signal(3).

ERANGE and EDOM are defined in <errno.h>. See intro(2) for explanation of EDOM and ERANGE.

NAME

initgroups - initialize group access list

SYNOPSIS

```
initgroups(name, basegid)
char *name;
int basegid;
```

DESCRIPTION

Initgroups reads through the group file and sets up, using the setgroups(2) call, the group access list for the user specified in name. The basegid is automatically included in the groups list. Typically this value is given as the group number from the password file.

FILES

/etc/group

SEE ALSO

setgroups(2)

DIAGNOSTICS

Initgroups returns -1 if it was not invoked by the super-user.

BUGS

Initgroups uses the routines based on getgrent(3). If the invoking program uses any of these routines, the group structure will be overwritten in the call to initgroups.

NAME

insque, remque - insert/remove element from a queue

SYNOPSIS

```
struct qelem {
   struct    qelem *q_forw;
   struct    qelem *q_back;
   char q_data[];
};

insque(elem, pred)
struct qelem *elem, *pred;

remque(elem)
struct qelem *elem;
```

DESCRIPTION

Insque and remque manipulate queues built from doubly linked lists. Each element in the queue must in the form of ``struct qelem''. Insque inserts elem in a queue immediately after pred; remque removes an entry elem from a queue.

SEE ALSO

``VAX Architecture Handbook'', pp. 228-235.

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

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

On a VAX, negative arguments cause y0, y1 and yn to return the reserved operand and set errno to EDOM.

SEE ALSO

math(3M), infnan(3M)

NAME

ldfps - load floating point status register

SYNOPSIS

```
void ldfps(fps)
unsigned int fps;
```

DESCRIPTION

Ldfps loads the hardware floating point status register with fps. See the PDP-11 hardware handbook for a description of the meaning of the various bits in fps.

BUGS

Ldfps is unique to the PDP-11 and 2BSD; its use is discouraged.

NAME
     lgamma - log gamma function

SYNOPSIS
```
     #include <math.h>

     double lgamma(x)
     double x;
```

DESCRIPTION
     Lgamma returns ln|I(x)|.

     The external integer signgam returns the sign of I(x) .

IDIOSYNCRASIES
     Do not use the expression signgam*exp(lgamma(x)) to compute g := I(x). Instead use a program like this (in C):
```
        lg = lgamma(x); g = signgam*exp(lg);
```

     Only after lgamma has returned can signgam be correct. Note too that I(x) must overflow when x is large enough, underflow when -x is large enough, and spawn a division by zero when x is a nonpositive integer.

     Only in the UNIX math library for C was the name gamma ever attached to lnI. Elsewhere, for instance in IBM's FORTRAN library, the name GAMMA belongs to I and the name ALGAMA to lnI in single precision; in double the names are DGAMMA and DLGAMA. Why should C be different?

     Archaeological records suggest that C's gamma originally delivered ln(I(|x|)). Later, the program gamma was changed to cope with negative arguments x in a more conventional way, but the documentation did not reflect that change correctly. The most recent change corrects inaccurate values when x is almost a negative integer, and lets I(x) be computed without conditional expressions. Programmers should not assume that lgamma has settled down.

     At some time in the future, the name gamma will be rehabilitated and used for the gamma function, just as is done in FORTRAN. The reason for this is not so much compatibility with FORTRAN as a desire to achieve greater speed for smaller values of |x| and greater accuracy for larger values.

     Meanwhile, programmers who have to use the name gamma in its former sense, for what is now lgamma, have two choices:

     1) Use the old math library, libom.

     2) Add the following program to your others:

```
      #include <math.h>
      double gamma(x)
      double x;
      {
           return (lgamma(x));
      }
```

DIAGNOSTICS

The reserved operand is returned on a VAX for negative integer arguments, errno is set to ERANGE; for very large arguments over/underflows will occur inside the lgamma routine.

SEE ALSO

math(3M), infnan(3M)

NAME

lib2648 - subroutines for the HP 2648 graphics terminal

SYNOPSIS

```
#include <stdio.h>

typedef char *bitmat;
FILE *trace;

cc file.c -l2648
```

DESCRIPTION

Lib2648 is a general purpose library of subroutines useful for interactive graphics on the Hewlett-Packard 2648 graphics terminal. To use it you must call the routine ttyinit() at the beginning of execution, and done() at the end of execution. All terminal input and output must go through the routines rawchar, readline, outchar, and outstr.

Lib2648 does the necessary ^E/^F handshaking if getenv(``TERM'') returns ``hp2648'', as it will if set by tset(1). Any other value, including for example ``2648'', will disable handshaking.

Bit matrix routines are provided to model the graphics memory of the 2648. These routines are generally useful, but are specifically useful for the update function which efficiently changes what is on the screen to what is supposed to be on the screen. The primative bit matrix routines are newmat, mat, and setmat.

The file trace, if non-null, is expected to be a file descriptor as returned by fopen. If so, lib2648 will trace the progress of the output by writing onto this file. It is provided to make debugging output feasible for graphics programs without messing up the screen or the escape sequences being sent. Typical use of trace will include:

```
switch (argv[1][1]) {
case 'T':
     trace = fopen("trace", "w");
     break;
...
if (trace)
     fprintf(trace, "x is %d, y is %s\n", x, y);
...
dumpmat("before update", xmat);
```

ROUTINES

agoto(x, y)
: Move the alphanumeric cursor to position (x, y), measured from the upper left corner of the screen.

aoff()
  Turn the alphanumeric display off.

aon()
  Turn the alphanumeric display on.

areaclear(rmin, cmin, rmax, cmax)
  Clear the area on the graphics screen bordered by the four arguments. In normal mode the area is set to all black, in inverse video mode it is set to all white.

beep()
  Ring the bell on the terminal.

bitcopy(dest, src, rows, cols) bitmat dest,
  Copy a rows by cols bit matrix from src to (user provided) dest.

cleara()
  Clear the alphanumeric display.

clearg()
  Clear the graphics display. Note that the 2648 will only clear the part of the screen that is visible if zoomed in.

curoff()
  Turn the graphics cursor off.

curon()
  Turn the graphics cursor on.

dispmsg(str, x, y, maxlen) char *str;
  Display the message str in graphics text at position (x, y). The maximum message length is given by maxlen, and is needed for dispmsg to know how big an area to clear before drawing the message. The lower left corner of the first character is at (x, y).

done()
  Should be called before the program exits. Restores the tty to normal, turns off graphics screen, turns on alphanumeric screen, flushes the standard output, etc.

draw(x, y)
  Draw a line from the pen location to (x, y). As with all graphics coordinates, (x, y) is measured from the bottom left corner of the screen. (x, y) coordinates represent the first quadrant of the usual Cartesian system.

drawbox(r, c, color, rows, cols)

Draw a rectangular box on the graphics screen. The lower left corner is at location (r, c). The box is rows rows high and cols columns wide. The box is drawn if color is 1, erased if color is 0. (r, c) absolute coordinates represent row and column on the screen, with the origin at the lower left. They are equivalent to (x, y) except for being reversed in order.

dumpmat(msg, m, rows, cols) char *msg; bitmat m;
   If trace is non-null, write a readable ASCII representation of the matrix m on trace. Msg is a label to identify the output.

emptyrow(m, rows, cols, r) bitmat m;
   Returns 1 if row r of matrix m is all zero, else returns 0. This routine is provided because it can be implemented more efficiently with a knowledge of the internal representation than a series of calls to mat.

error(msg) char *msg;
   Default error handler. Calls message(msg) and returns. This is called by certain routines in lib2648. It is also suitable for calling by the user program. It is probably a good idea for a fancy graphics program to supply its own error procedure which uses setjmp(3) to restart the program.

gdefault()
   Set the terminal to the default graphics modes.

goff()
   Turn the graphics display off.

gon()
   Turn the graphics display on.

koff()
   Turn the keypad off.

kon()
   Turn the keypad on. This means that most special keys on the terminal (such as the alphanumeric arrow keys) will transmit an escape sequence instead of doing their function locally.

line(x1, y1, x2, y2)
   Draw a line in the current mode from (x1, y1) to (x2, y2). This is equivalent to move(x1, y1); draw(x2, y2); except that a bug in the terminal involving repeated lines from the same point is compensated for.

lowleft()

Move the alphanumeric cursor to the lower left (home down) position.

mat(m, rows, cols, r, c) bitmat m;
Used to retrieve an element from a bit matrix. Returns 1 or 0 as the value of the [r, c] element of the rows by cols matrix m. Bit matrices are numbered (r, c) from the upper left corner of the matrix, beginning at (0, 0). R represents the row, and c represents the column.

message(str) char *str;
Display the text message str at the bottom of the graphics screen.

minmax(g, rows, cols, rmin, cmin, rmax, cmax) bitmat g; int *rmin, *cmin, *rmax, *cmax;
Find the smallest rectangle that contains all the 1 (on) elements in the bit matrix g. The coordinates are returned in the variables pointed to by rmin, cmin, rmax, cmax.

move(x, y)
Move the pen to location (x, y). Such motion is internal and will not cause output until a subsequent sync().

movecurs(x, y)
Move the graphics cursor to location (x, y).

bitmat newmat(rows, cols)
Create (with malloc(3)) a new bit matrix of size rows by cols. The value created (e.g. a pointer to the first location) is returned. A bit matrix can be freed directly with free.

outchar(c) char c;
Print the character c on the standard output. All output to the terminal should go through this routine or outstr.

outstr(str) char *str;
Print the string str on the standard output by repeated calls to outchar.

printg()
Print the graphics display on the printer. The printer must be configured as device 6 (the default) on the HPIB.

char rawchar()
Read one character from the terminal and return it. This routine or readline should be used to get all

input, rather than getchar(3).

rboff()
Turn the rubber band line off.

rbon()
Turn the rubber band line on.

char *rdchar(c) char c;
Return a readable representation of the character c. If c is a printing character it returns itself, if a control character it is shown in the ^X notation, if negative an apostrophe is prepended. Space returns ^`, rubout returns ^?.

NOTE: A pointer to a static place is returned. For this reason, it will not work to pass rdchar twice to the same fprintf/sprintf call. You must instead save one of the values in your own buffer with strcpy.

readline(prompt, msg, maxlen) char *prompt, *msg;
Display prompt on the bottom line of the graphics display and read one line of text from the user, terminated by a newline. The line is placed in the buffer msg, which has size maxlen characters. Backspace processing is supported.

setclear()
Set the display to draw lines in erase mode. (This is reversed by inverse video mode.)

setmat(m, rows, cols, r, c, val) bitmat m;
The basic operation to store a value in an element of a bit matrix. The [r, c] element of m is set to val, which should be either 0 or 1.

setset()
Set the display to draw lines in normal (solid) mode. (This is reversed by inverse video mode.)

setxor()
Set the display to draw lines in exclusive or mode.

sync()
Force all accumulated output to be displayed on the screen. This should be followed by fflush(stdout). The cursor is not affected by this function. Note that it is normally never necessary to call sync, since rawchar and readline call sync() and fflush(stdout) automatically.

togvid()

Toggle the state of video. If in normal mode, go into inverse video mode, and vice versa. The screen is reversed as well as the internal state of the library.

ttyinit()
Set up the terminal for processing. This routine should be called at the beginning of execution. It places the terminal in CBREAK mode, turns off echo, sets the proper modes in the terminal, and initializes the library.

update(mold, mnew, rows, cols, baser, basec) bitmat mold, mnew;
Make whatever changes are needed to make a window on the screen look like mnew. Mold is what the window on the screen currently looks like. The window has size rows by cols, and the lower left corner on the screen of the window is [baser, basec]. Note: update was not intended to be used for the entire screen. It would work but be very slow and take 64K bytes of memory just for mold and mnew. It was intended for 100 by 100 windows with objects in the center of them, and is quite fast for such windows.

vidinv()
Set inverse video mode.

vidnorm()
Set normal video mode.

zermat(m, rows, cols) bitmat m;
Set the bit matrix m to all zeros.

zoomn(size)
Set the hardware zoom to value size, which can range from 1 to 15.

zoomoff()
Turn zoom off. This forces the screen to zoom level 1 without affecting the current internal zoom number.

zoomon()
Turn zoom on. This restores the screen to the previously specified zoom size.

DIAGNOSTICS

The routine error is called when an error is detected. The only error currently detected is overflow of the buffer provided to readline.

Subscripts out of bounds to setmat return without setting anything.

FILES
     /usr/lib/lib2648.a

SEE ALSO
     fed(1)

AUTHOR
     Mark Horton

BUGS
     This library is not supported. It makes no attempt to use all of the features of the terminal, only those needed by fed. Contributions from users will be accepted for addition to the library.

     The HP 2648 terminal is somewhat unreliable at speeds over 2400 baud, even with the ^E/^F handshaking. In an effort to improve reliability, handshaking is done every 32 characters. (The manual claims it is only necessary every 80 characters.) Nonetheless, I/O errors sometimes still occur.

     There is no way to control the amount of debugging output generated on trace without modifying the source to the library.

NAME

malloc, free, realloc, calloc, alloca - memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

char *alloca(size)
int size;
```

DESCRIPTION

Malloc and free provide a general-purpose memory allocation package. Malloc returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free is a pointer to a block previously allocated by malloc; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc maintains multiple lists of free blocks according to size, allocating space from the appropriate list. It calls sbrk (see brk(2)) to get more memory from the system when there is no suitable space already free.

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.

In order to be compatible with older versions, realloc also works if ptr points to a block freed since the last call of malloc, realloc or calloc; sequences of free, malloc and realloc were previously used to attempt storage compaction. This procedure is no longer recommended.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Alloca allocates size bytes of space in the stack frame of the caller. This temporary space is automatically freed on return.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object. If the space is of pagesize or larger, the memory returned will be page-aligned.

SEE ALSO

brk(2), pagesize(2)

DIAGNOSTICS

Malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. Malloc may be recompiled to check the arena very stringently on every transaction; those sites with a source code license may check the source code to see how this can be done.

BUGS

When realloc returns 0, the block pointed to by ptr may be destroyed.

The current implementation of malloc does not always fail gracefully when system memory limits are approached. It may fail to allocate memory when larger free blocks could be broken up, or when limits are exceeded because the size is rounded up. It is optimized for sizes that are powers of two.

Alloca is machine dependent; its use is discouraged.

NAME

math - introduction to mathematical library functions

DESCRIPTION

These functions constitute the C math library, libm. The link editor searches this library under the "-lm" option. Declarations for these functions may be obtained from the include file <math.h>. The Fortran math library is described in ``man 3f intro''.

LIST OF FUNCTIONS

Name	Appears on Page	Description	Error Bound (ULPs)
acos	sin.3m	inverse trigonometric function	3
acosh	asinh.3m	inverse hyperbolic function	3
asin	sin.3m	inverse trigonometric function	3
asinh	asinh.3m	inverse hyperbolic function	3
atan	sin.3m	inverse trigonometric function	1
atanh	asinh.3m	inverse hyperbolic function	3
atan2	sin.3m	inverse trigonometric function	2
cabs	hypot.3m	complex absolute value	1
cbrt	sqrt.3m	cube root	1
ceil	floor.3m	integer no less than	0
copysign	ieee.3m	copy sign bit	0
cos	sin.3m	trigonometric function	1
cosh	sinh.3m	hyperbolic function	3
drem	ieee.3m	remainder	0
erf	erf.3m	error function	???
erfc	erf.3m	complementary error function	???
exp	exp.3m	exponential	1
expm1	exp.3m	exp(x)-1	1
fabs	floor.3m	absolute value	0
floor	floor.3m	integer no greater than	0
hypot	hypot.3m	Euclidean distance	1
infnan	infnan.3m	signals exceptions	
j0	j0.3m	bessel function	???
j1	j0.3m	bessel function	???
jn	j0.3m	bessel function	???
lgamma	lgamma.3m	log gamma function; (formerly gamma.3m)	
log	exp.3m	natural logarithm	1
logb	ieee.3m	exponent extraction	0
log10	exp.3m	logarithm to base 10	3
log1p	exp.3m	log(1+x)	1
pow	exp.3m	exponential x**y	60-500
rint	floor.3m	round to nearest integer	0
scalb	ieee.3m	exponent adjustment	0
sin	sin.3m	trigonometric function	1
sinh	sinh.3m	hyperbolic function	3
sqrt	sqrt.3m	square root	1
tan	sin.3m	trigonometric function	3
tanh	sinh.3m	hyperbolic function	3
y0	j0.3m	bessel function	???
y1	j0.3m	bessel function	???

yn	j0.3m	bessel function	???

NOTES

In 4.3 BSD, distributed from the University of California in late 1985, most of the foregoing functions come in two versions, one for the double-precision "D" format in the DEC VAX-11 family of computers, another for double-precision arithmetic conforming to the IEEE Standard 754 for Binary Floating-Point Arithmetic. The two versions behave very similarly, as should be expected from programs more accurate and robust than was the norm when UNIX was born. For instance, the programs are accurate to within the numbers of ulps tabulated above; an ulp is one Unit in the Last Place. And the programs have been cured of anomalies that afflicted the older math library libm in which incidents like the following had been reported:

```
   sqrt(-1.0) = 0.0 and log(-1.0) = -1.7e38.
   cos(1.0e-11) > cos(0.0) > 1.0.
   pow(x,1.0) != x when x = 2.0, 3.0, 4.0, ..., 9.0.
   pow(-1.0,1.0e10) trapped on Integer Overflow.
   sqrt(1.0e30) and sqrt(1.0e-30) were very slow.
```

However the two versions do differ in ways that have to be explained, to which end the following notes are provided.

DEC VAX-11 D_floating-point:

This is the format for which the original math library libm was developed, and to which this manual is still principally dedicated. It is the double-precision format for the PDP-11 and the earlier VAX-11 machines; VAX-11s after 1983 were provided with an optional "G" format closer to the IEEE double-precision format. The earlier DEC MicroVAXs have no D format, only G double-precision. (Why? Why not?)

```
Properties of D_floating-point:
   Wordsize: 64 bits, 8 bytes.  Radix: Binary.
   Precision: 56 sig.  bits, roughly like 17 sig.
   decimals.
        If x and x' are consecutive positive
        D_floating-point numbers (they differ by 1 ulp),
        then
        1.3e-17 < 0.5**56 < (x'-x)/x < 0.5**55 < 2.8e-17.
   Range: Overflow threshold  = 2.0**127 = 1.7e38.
        Underflow threshold = 0.5**128 = 2.9e-39.
        NOTE:     THIS RANGE IS COMPARATIVELY NARROW.
        Overflow customarily stops computation.
        Underflow is customarily flushed quietly to zero.
        CAUTION:
           It is possible to have x != y and yet x-y = 0
           because of underflow.  Similarly x > y > 0
           cannot prevent either x*y = 0 or  y/x = 0
           from happening without warning.
```

Zero is represented ambiguously.
Although 2**55 different representations of zero are accepted by the hardware, only the obvious representation is ever produced. There is no -0 on a VAX.

Infinity is not part of the VAX architecture.

Reserved operands:
of the 2**55 that the hardware recognizes, only one of them is ever produced. Any floating-point operation upon a reserved operand, even a MOVF or MOVD, customarily stops computation, so they are not much used.

Exceptions:
Divisions by zero and operations that overflow are invalid operations that customarily stop computation or, in earlier machines, produce reserved operands that will stop computation.

Rounding:
Every rational operation (+, -, *, /) on a VAX (but not necessarily on a PDP-11), if not an over/underflow nor division by zero, is rounded to within half an ulp, and when the rounding error is exactly half an ulp then rounding is away from 0.

Except for its narrow range, D_floating-point is one of the better computer arithmetics designed in the 1960's. Its properties are reflected fairly faithfully in the elementary functions for a VAX distributed in 4.3 BSD. They over/underflow only if their results have to lie out of range or very nearly so, and then they behave much as any rational arithmetic operation that over/underflowed would behave. Similarly, expressions like log(0) and atanh(1) behave like 1/0; and sqrt(-3) and acos(3) behave like 0/0; they all produce reserved operands and/or stop computation! The situation is described in more detail in manual pages.

This response seems excessively punitive, so it is destined to be replaced at some time in the foreseeable future by a more flexible but still uniform scheme being developed to handle all floating-point arithmetic exceptions neatly. See infnan(3M) for the present state of affairs.

How do the functions in 4.3 BSD's new libm for UNIX compare with their counterparts in DEC's VAX/VMS library? Some of the VMS functions are a little faster, some are a little more accurate, some are more puritanical about exceptions (like pow(0.0,0.0) and atan2(0.0,0.0)), and most occupy much more memory than their counterparts in libm. The VMS codes interpolate in large table to achieve speed and accuracy; the libm codes use tricky formulas compact enough that all of them may some day fit into a ROM.

More important, DEC regards the VMS codes as proprietary and guards them zealously against unauthorized use. But the libm codes in 4.3 BSD are intended for the public domain; they may be copied freely provided their provenance is always acknowledged, and provided users assist the authors in their researches by reporting experience with the codes. Therefore no user of UNIX on a machine whose arithmetic resembles VAX D_floating-point need use anything worse than the new libm.

IEEE STANDARD 754 Floating-Point Arithmetic:

This standard is on its way to becoming more widely adopted than any other design for computer arithmetic. VLSI chips that conform to some version of that standard have been produced by a host of manufacturers, among them ...

```
   Intel i8087, i80287     National Semiconductor  32081
   Motorola 68881    Weitek WTL-1032, ... , -1165
   Zilog Z8070             Western Electric (AT&T) WE32106.
```

Other implementations range from software, done thoroughly in the Apple Macintosh, through VLSI in the Hewlett-Packard 9000 series, to the ELXSI 6400 running ECL at 3 Megaflops. Several other companies have adopted the formats of IEEE 754 without, alas, adhering to the standard's way of handling rounding and exceptions like over/underflow. The DEC VAX G_floating-point format is very similar to the IEEE 754 Double format, so similar that the C programs for the IEEE versions of most of the elementary functions listed above could easily be converted to run on a MicroVAX, though nobody has volunteered to do that yet.

The codes in 4.3 BSD's libm for machines that conform to IEEE 754 are intended primarily for the National Semi. 32081 and WTL 1164/65. To use these codes with the Intel or Zilog chips, or with the Apple Macintosh or ELXSI 6400, is to forego the use of better codes provided (perhaps freely) by those companies and designed by some of the authors of the codes above. Except for atan, cabs, cbrt, erf, erfc, hypot, j0-jn, lgamma, pow and y0-yn, the Motorola 68881 has all the functions in libm on chip, and faster and more accurate; it, Apple, the i8087, Z8070 and WE32106 all use 64 sig. bits. The main virtue of 4.3 BSD's libm codes is that they are intended for the public domain; they may be copied freely provided their provenance is always acknowledged, and provided users assist the authors in their researches by reporting experience with the codes. Therefore no user of UNIX on a machine that conforms to IEEE 754 need use anything worse than the new libm.

Properties of IEEE 754 Double-Precision:
   Wordsize: 64 bits, 8 bytes. Radix: Binary.
   Precision: 53 sig. bits, roughly like 16 sig.

```
decimals.
     If x and x' are consecutive positive
     Double-Precision numbers (they differ by 1 ulp),
     then
     1.1e-16 < 0.5**53 < (x'-x)/x < 0.5**52 < 2.3e-16.
Range: Overflow threshold  = 2.0**1024 = 1.8e308
     Underflow threshold = 0.5**1022 = 2.2e-308
     Overflow goes by default to a signed Infinity.
     Underflow is Gradual, rounding to the nearest
     integer multiple of 0.5**1074 = 4.9e-324.
Zero is represented ambiguously as +0 or -0.
     Its sign transforms correctly through multiplica-
     tion or division, and is preserved by addition of
     zeros with like signs; but x-x yields +0 for every
     finite x.  The only operations that reveal zero's
     sign are division by zero and copysign(x,+0).  In
     particular, comparison (x > y, x > y, etc.) cannot
     be affected by the sign of zero; but if finite x =
     y then Infinity = 1/(x-y) != -1/(y-x) = -Infinity.
Infinity is signed.
     it persists when added to itself or to any finite
     number.   Its sign transforms correctly through
     multiplication and division, and
     (finite)/+Infinity = +0 (nonzero)/0 = +Infinity.
     But Infinity-Infinity, Infinity*0 and
     Infinity/Infinity are, like 0/0 and sqrt(-3),
     invalid operations that produce NaN. ...
Reserved operands:
     there are 2**53-2 of them, all called NaN (Not a
     Number).  Some, called Signaling NaNs, trap any
     floating-point operation performed upon them; they
     are used to mark missing or uninitialized values,
     or nonexistent elements of arrays.  The rest are
     Quiet NaNs; they are the default results of
     Invalid Operations, and propagate through subse-
     quent arithmetic operations.  If x != x then x is
     NaN; every other predicate (x > y, x = y, x < y,
     ...) is FALSE if NaN is involved.
     NOTE: Trichotomy is violated by NaN.
        Besides being FALSE, predicates that entail
        ordered comparison, rather than mere
        (in)equality, signal Invalid Operation when
        NaN is involved.
Rounding:
     Every algebraic operation (+, -, *, /, sqrt) is
     rounded by default to within half an ulp, and when
     the rounding error is exactly half an ulp then the
     rounded value's least significant bit is zero.
     This kind of rounding is usually the best kind,
     sometimes provably so; for instance, for every x =
     1.0, 2.0, 3.0, 4.0, ..., 2.0**52, we find
     (x/3.0)*3.0 == x and (x/10.0)*10.0 == x and ...
```

despite that both the quotients and the products have been rounded. Only rounding like IEEE 754 can do that. But no single kind of rounding can be proved best for every circumstance, so IEEE 754 provides rounding towards zero or towards +Infinity or towards -Infinity at the programmer's option.   And the same kinds of rounding are specified for Binary-Decimal Conversions, at least for magnitudes between roughly 1.0e-10 and 1.0e37.

Exceptions:

IEEE 754 recognizes five kinds of floating-point exceptions, listed below in declining order of probable importance.

Exception	Default Result
Invalid Operation	NaN, or FALSE
Overflow	+Infinity
Divide by Zero	+Infinity
Underflow	Gradual Underflow
Inexact	Rounded value

NOTE: An Exception is not an Error unless handled badly. What makes a class of exceptions exceptional is that no single default response can be satisfactory in every instance. On the other hand, if a default response will serve most instances satisfactorily, the unsatisfactory instances cannot justify aborting computation every time the exception occurs.

For each kind of floating-point exception, IEEE 754 provides a Flag that is raised each time its exception is signaled, and stays raised until the program resets it. Programs may also test, save and restore a flag. Thus, IEEE 754 provides three ways by which programs may cope with exceptions for which the default result might be unsatisfactory:

1) Test for a condition that might cause an exception later, and branch to avoid the exception.

2) Test a flag to see whether an exception has occurred since the program last reset its flag.

3) Test a result to see whether it is a value that only an exception could have produced.
    CAUTION: The only reliable ways to discover whether Underflow has occurred are to test whether products or quotients lie closer to zero than the underflow threshold, or to test the Underflow flag. (Sums and differences cannot underflow in IEEE 754; if x != y then x-y is correct to full precision and certainly nonzero regardless of how tiny it may be.)

Products and quotients that underflow gradually can lose accuracy gradually without vanishing, so comparing them with zero (as one might on a VAX) will not reveal the loss. Fortunately, if a gradually underflowed value is destined to be added to something bigger than the underflow threshold, as is almost always the case, digits lost to gradual underflow will not be missed because they would have been rounded off anyway. So gradual underflows are usually provably ignorable. The same cannot be said of underflows flushed to 0.

At the option of an implementor conforming to IEEE 754, other ways to cope with exceptions may be provided:

4) ABORT. This mechanism classifies an exception in advance as an incident to be handled by means traditionally associated with error-handling statements like "ON ERROR GO TO ...". Different languages offer different forms of this statement, but most share the following characteristics:

- No means is provided to substitute a value for the offending operation's result and resume computation from what may be the middle of an expression. An exceptional result is abandoned.

- In a subprogram that lacks an error-handling statement, an exception causes the subprogram to abort within whatever program called it, and so on back up the chain of calling subprograms until an error-handling statement is encountered or the whole task is aborted and memory is dumped.

5) STOP. This mechanism, requiring an interactive debugging environment, is more for the programmer than the program. It classifies an exception in advance as a symptom of a programmer's error; the exception suspends execution as near as it can to the offending operation so that the programmer can look around to see how it happened. Quite often the first several exceptions turn out to be quite unexceptionable, so the programmer ought ideally to be able to resume execution after each one as if execution had not been stopped.

6) ... Other ways lie beyond the scope of this document.

The crucial problem for exception handling is the problem of Scope, and the problem's solution is understood, but not enough manpower was available to implement it fully in time to be distributed in 4.3 BSD's libm. Ideally, each elementary function should act as if it were indivisible, or atomic, in the sense that ...

i) No exception should be signaled that is not deserved by the data supplied to that function.

ii) Any exception signaled should be identified with that function rather than with one of its subroutines.

iii) The internal behavior of an atomic function should not be disrupted when a calling program changes from one to another of the five or so ways of handling exceptions listed above, although the definition of the function may be correlated intentionally with exception handling.

Ideally, every programmer should be able conveniently to turn a debugged subprogram into one that appears atomic to its users. But simulating all three characteristics of an atomic function is still a tedious affair, entailing hosts of tests and saves-restores; work is under way to ameliorate the inconvenience.

Meanwhile, the functions in libm are only approximately atomic. They signal no inappropriate exception except possibly ...

Over/Underflow
    when a result, if properly computed, might have lain barely within range, and

Inexact in cabs, cbrt, hypot, log10 and pow
    when it happens to be exact, thanks to fortuitous cancellation of errors.

Otherwise, ...

Invalid Operation is signaled only when
    any result but NaN would probably be misleading.

Overflow is signaled only when
    the exact result would be finite but beyond the overflow threshold.

Divide-by-Zero is signaled only when
    a function takes exactly infinite values at finite operands.

Underflow is signaled only when
    the exact result would be nonzero but tinier than the underflow threshold.

Inexact is signaled only when
    greater range or precision would be needed to represent the exact result.

BUGS

When signals are appropriate, they are emitted by certain operations within the codes, so a subroutine-trace may be needed to identify the function with its signal in case method 5) above is in use. And the codes all take the IEEE 754 defaults for granted; this means that a decision to trap all divisions by zero could disrupt a code that would otherwise get correct results despite division by zero.

SEE ALSO

An explanation of IEEE 754 and its proposed extension p854 was published in the IEEE magazine MICRO in August 1984 under the title "A Proposed Radix- and Word-length-independent Standard for Floating-point Arithmetic" by W. J. Cody et al. The manuals for Pascal, C and BASIC on the Apple Macintosh document the features of IEEE 754 pretty well. Articles in the IEEE magazine COMPUTER vol. 14 no. 3 (Mar. 1981), and in the ACM SIGNUM Newsletter Special Issue of Oct. 1979, may be helpful although they pertain to superseded drafts of the standard.

AUTHOR

W. Kahan, with the help of Z-S. Alex Liu, Stuart I. McDonald, Dr. Kwok-Choi Ng, Peter Tang.

NAME
     mktemp - make a unique file name

SYNOPSIS
```
     char *mktemp(template)
     char *template;

     mkstemp(template)
     char *template;
```

DESCRIPTION
     Mktemp creates a unique file name and returns the address of the template. The template should contain a file name with trailing X's, normally something like /tmp/tempXXXXXX. The X's are replaced by the current process number and/or a unique letter combination. Mkstemp makes the same replacement to the template but opens the template file and returns a file descriptor open for reading and writing. Mkstemp avoids the race between testing whether the file exists and opening it for use. The number of file name combinations mktemp and mkstemp will try depends on the number of X's placed on the end of the template; six X's will result in them trying roughly 26 ** 6 combinations.

SEE ALSO
     access(2), getpid(2), open(2)

DIAGNOSTICS
     Mktemp returns NULL on failure, mkstemp returns -1 if no suitable file could be created.

NAME

monitor, monstartup, moncontrol - prepare execution profile

SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)
```

DESCRIPTION

There are two different forms of monitoring available: An executable program created by:

```
   cc -p . . .
```

automatically includes calls for the prof(1) monitor and includes an initial call to its start-up routine monstartup with default parameters; monitor need not be called explicitly except to gain fine control over profil buffer allocation. An executable program created by:

```
   cc -pg . . .
```

automatically includes calls for the gprof(1) monitor.

Monstartup is a high level interface to profil(2). Lowpc and highpc specify the address range that is to be sampled; the lowest address sampled is that of lowpc and the highest is just below highpc. Monstartup allocates space using sbrk(2) and passes it to monitor (see below) to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. Only calls of functions compiled with the profiling option -p of cc(1) are recorded.

To profile the entire program, it is sufficient to use

```
   extern etext();
   . . .
   monstartup((int) 2, etext);
```

Etext lies just above all the program text, see end(3).

To stop execution monitoring and write the results on the file mon.out, use

```
   monitor(0);
```

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

Moncontrol is used to selectively control profiling within a program. This works with either prof(1) or gprof(1) type profiling. When the program starts, profiling begins. To stop the collection of histogram ticks and call counts use moncontrol(0); to resume the collection of histogram ticks and call counts use moncontrol(1). This allows the cost of particular operations to be measured. Note that an output file will be produced upon program exit irregardless of the state of moncontrol.

Monitor is a low level interface to profil(2). Lowpc and highpc are the addresses of two functions; buffer is the address of a (user supplied) array of bufsize short integers. At most nfunc call counts can be kept. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. Monitor divides the buffer into space to record the histogram of program counter samples over the range lowpc to highpc, and space to record call counts of functions compiled with the -p option to cc(1).

To profile the entire program, it is sufficient to use

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

FILES
mon.out

SEE ALSO
cc(1), prof(1), gprof(1), profil(2), sbrk(2)

NAME

madd, msub, mult, mdiv, pow, gcd, invert, rpow, msqrt, mcmp, move, min, omin, fmin, m_in, mout, omout, fmout, m_out, sdiv, itom - multiple precision integer arithmetic

SYNOPSIS

```
#include <mp.h>
#include <stdio.h>

typedef struct mint { int len; short *val; } MINT;

madd(a, b, c)
msub(a, b, c)
mult(a, b, c)
mdiv(a, b, q, r)
pow(a, b, m, c)
gcd(a, b, c)
invert(a, b, c)
rpow(a, n, c)
msqrt(a, b, r)
mcmp(a, b)
move(a, b)
min(a)
omin(a)
fmin(a, f)
m_in(a, n, f)
mout(a)
omout(a)
fmout(a, f)
m_out(a, n, f)
MINT *a, *b, *c, *m, *q, *r;
FILE *f;
int n;

sdiv(a, n, q, r)
MINT *a, *q;
short n;
short *r;

MINT *itom(n)
```

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type MINT. Pointers to a MINT can be initialized using the function itom which sets the initial value to n. After that, space is managed automatically by the routines.

madd, msub and mult assign to c the sum, difference and product, respectively, of a and b. mdiv assigns to q and r the quotient and remainder obtained from dividing a by b. sdiv is like mdiv except that the divisor is a short integer n and the remainder is placed in a short whose address is given as r. msqrt produces the integer square root of a in b and places the remainder in r. rpow calculates in c the value of a raised to the (``regular'' integral) power n, while pow calculates this with a full multiple precision exponent b and the result is reduced modulo m. gcd returns the greatest common denominator of a and b in c, and invert computes c such that a*c mod b = 1, for a and b relatively prime. mcmp returns a negative, zero or positive integer value when a is less than, equal to or greater than b, respectively. move copies a to b. min and mout do decimal input and output while omin and omout do octal input and output. More generally, fmin and fmout do decimal input and output using file f, and m_in and m_out do I/O with arbitrary radix n. On input, records should have the form of strings of digits terminated by a newline; output records have a similar form.

Programs which use the multiple-precision arithmetic library must be loaded using the loader flag -lmp.

FILES

/usr/include/mp.h	include file
/usr/lib/libmp.a	object code library

SEE ALSO

dc(1), bc(1)

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

BUGS

Bases for input and output should be <= 10.

dc(1) and bc(1) don't use this library.

The input and output routines are a crock.

pow is also the name of a standard math library routine.

NAME
     dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete,
     dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr - data
     base subroutines

SYNOPSIS
```
     #include <ndbm.h>

     typedef struct {
       char *dptr;
       int dsize;
     } datum;

     DBM *dbm_open(file, flags, mode)
       char *file;
       int flags, mode;

     void dbm_close(db)
       DBM *db;

     datum dbm_fetch(db, key)
       DBM *db;
       datum key;

     int dbm_store(db, key, content, flags)
       DBM *db;
       datum key, content;
       int flags;

     int dbm_delete(db, key)
       DBM *db;
       datum key;

     datum dbm_firstkey(db)
       DBM *db;

     datum dbm_nextkey(db)
       DBM *db;

     int dbm_error(db)
       DBM *db;

     int dbm_clearerr(db)
       DBM *db;
```

DESCRIPTION
     These functions maintain key/content pairs in a data base.
     The functions will handle very large (a billion blocks)
     databases and will access a keyed item in one or two file
     system accesses. This package replaces the earlier dbm(3x)
     library, which managed only a single database.

Keys and contents are described by the datum typedef. A datum specifies a string of dsize bytes pointed to by dptr. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has `.dir' as its suffix. The second file contains all data and has `.pag' as its suffix.

Before a database can be accessed, it must be opened by dbm_open. This will open and/or create the files file.dir and file.pag depending on the flags parameter (see open(2)).

Once open, the data stored under a key is accessed by dbm_fetch and data is placed under a key by dbm_store. The flags field can be either DBM_INSERT or DBM_REPLACE. DBM_INSERT will only insert new entries into the database and will not change an existing entry with the same key. DBM_REPLACE will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by dbm_delete. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of dbm_firstkey and dbm_nextkey. Dbm_firstkey will return the first key in the database. Dbm_nextkey will return the next key in the database. This code will traverse the data base:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key =
dbm_nextkey(db))
```

Dbm_error returns non-zero when an error has occurred reading or writing the database. Dbm_clearerr resets the error condition on the named database.

DIAGNOSTICS

All functions that return an int indicate errors with negative values. A zero return indicates ok. Routines that return a datum indicate errors with a null (0) dptr. If dbm_store called with a flags value of DBM_INSERT finds an existing entry with the same key it returns 1.

BUGS

The `.pag' file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes). Moreover all key/content pairs that hash together must fit on a single block. Dbm_store will return an error in the event that a disk block fills with inseparable data.

Dbm_delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by dbm_firstkey and dbm_nextkey depends on a hashing function, not on anything interesting.

SEE ALSO

dbm(3X)

NAME
     nice - set program priority

SYNOPSIS
     nice(incr)

DESCRIPTION
     This interface is obsoleted by setpriority(2).

     The scheduling priority of the process is augmented by incr. Positive priorities get less service than normal.   Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration.

     Negative increments are ignored except on behalf of the super-user. The priority is limited to the range -20 (most urgent) to 20 (least).

     The priority of a process is passed to a child process by fork(2). For a privileged process to return to normal priority from an unknown state, nice should be called successively with arguments -40 (goes to priority -20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO
     nice(1), setpriority(2), fork(2), renice(8)

NAME

nlist - get entries from name list

SYNOPSIS

```
#include <nlist.h>

nlist(filename, nl)
char *filename;
struct nlist nl[];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See a.out(5) for the structure declaration.

This subroutine is useful for examining the system name list kept in the file /vmunix. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

If the file cannot be found or if it is not a valid namelist -1 is returned; otherwise, the number of unfound namelist entries is returned.

The type entry is set to 0 if the symbol is not found.

NAME

ns_addr, ns_ntoa - Xerox NS(tm) address conversion routines

SYNOPSIS

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

struct ns_addr ns_addr(cp)
char *cp;

char *ns_ntoa(ns)
struct ns_addr ns;
```

DESCRIPTION

The routine ns_addr interprets character strings representing XNS addresses, returning binary information suitable for use in system calls. ns_ntoa takes XNS addresses and returns ASCII strings representing the address in a notation in common use in the Xerox Development Environment:

```
   <network number>.<host number>.<port number>
```

Trailing zero fields are suppressed, and each number is printed in hexadecimal, in a format suitable for input to ns_addr. Any fields lacking super-decimal digits will have a trailing ``H'' appended.

Unfortunately, no universal standard exists for representing XNS addresses. An effort has been made to insure that ns_addr be compatible with most formats in common use. It will first separate an address into 1 to 3 fields using a single delimiter chosen from period (``.''), colon (``:'') or pound-sign (``#''). Each field is then examined for byte separators (colon or period). If there are byte separators, each subfield separated is taken to be a small hexadecimal number, and the entirety is taken as a network-byte-ordered quantity to be zero extended in the high-network-order bytes. Next, the field is inspected for hyphens, in which case the field is assumed to be a number in decimal notation with hyphens separating the millenia. Next, the field is assumed to be a number: It is interpreted as hexadecimal if there is a leading ``0x'' (as in C), a trailing ``H'' (as in Mesa), or there are any super-decimal digits present. It is interpreted as octal is there is a leading ``0'' and there are no super-octal digits. Otherwise, it is converted as a decimal number.

SEE ALSO

hosts(5), networks(5),

DIAGNOSTICS

None (see BUGS).

BUGS

The string returned by ns_ntoa resides in a static memory area.
ns_addr should diagnose improperly formed input, and there should be an unambiguous way to recognize this.

NAME
     pause - stop until signal

SYNOPSIS
     pause()

DESCRIPTION
     Pause never returns normally. It is used to give up control while waiting for a signal from kill(2) or an interval timer, see setitimer(2). Upon termination of a signal handler started during a pause, the pause call will return.

RETURN VALUE
     Always returns -1.

ERRORS
     Pause always returns:

[EINTR]	The call was interrupted.

SEE ALSO
     kill(2), select(2), sigpause(2)

NAME
     perror, strerror - system error messages

SYNOPSIS
```
perror(s)
char *s;

#include <string.h>

char *
strerror(errnum)
int errnum;
```

DESCRIPTION

The strerror() and perror() functions look up the error message string corresponding to an error number.

The strerror() function accepts an error number argument errnum and returns a pointer to the corresponding message string.

The perror() function finds the error message corresponding to the current value of the global variable errno (intro(2)) and writes it, followed by a newline, to the standard error file descriptor. If the argument string is non-NULL, it is prepended to the message string and separated from it by a colon and space (`: '). If string is NULL, only the error message string is printed.

If errnum is not a recognized error number, the error message string will contain ``Unknown error: '' followed by the error number in decimal.

The error messages are stored in a data file now rather than an in memory array. See syserror(5).

SEE ALSO

mkerrlst(1), intro(2), psignal(3), strerror(3), syserror(3), syserror(5)

BUGS

The strerror() function returns its result in a static buffer which may be overwritten by subsequent calls.

The array sys_errlist[] and the global sys_nerr are obsolete and should not be used. They have, for the time being, been placed in an object library liberrlst.a.

NAME
     plot - graphics filters

SYNOPSIS
     plot [-Tterminal] [-rresolution] [files...]

DESCRIPTION
     These commands read plotting instructions (see plot(5)) from the standard input or the specified files, and in general produce plotting instructions suitable for a particular terminal on the standard output. The -r flag may be used to specify the device's output resolution (currently only the Imagen laser printer understands this option).

     If no terminal type is specified, the environment parameter $TERM (see environ(7)) is used. Known terminals are:

     4013 Tektronix 4013 storage scope.

     4014 or tek
        Tektronix 4014 or 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or 4015 without the Enhanced Graphics Module).

     450 DASI Hyterm 450 terminal (Diablo mechanism).

     300 DASI 300 or GSI terminal (Diablo mechanism).

     300S DASI 300S terminal (Diablo mechanism).

     aed AED 512 color graphics terminal.

     bitgraph or bg
        BBN bitgraph graphics terminal.

     imagen or ip
        Imagen laser printer (default 240 dots-per-inch resolution).

     crt Any crt terminal capable of running vi(1).

     dumb Dumb terminals without cursor addressing or line printers.

     vt125
        DEC vt125 terminal.

     hp2648 or hp or hp8
        Hewlett Packard 2648 graphics terminal.

     ver Versatec D1200A printer-plotter.

var Benson Varian printer-plotter.

These versions of plot use the -g option of lpr(1) to send the result directly to the plotter device rather than to the standard output.

FILES
```
     /usr/bin/t4013
     /usr/bin/tek
     /usr/bin/t450
     /usr/bin/t300
     /usr/bin/t300s
     /usr/bin/aedplot
     /usr/bin/bgplot
     /usr/bin/crtplot
     /usr/bin/dumbplot
     /usr/bin/gigiplot
     /usr/bin/hpplot
     /usr/bin/implot
     /usr/ucb/lpr
```

SEE ALSO
plot(3X), plot(3F), plot(5), lpr(1)

NAME
     plot: openpl, erase, label, line, circle, arc, move, cont,
     point, linemod, space, closepl - graphics interface

SYNOPSIS
```
     openpl()

     erase()

     label(s)
     char s[];

     line(x1, y1, x2, y2)

     circle(x, y, r)

     arc(x, y, x0, y0, x1, y1)

     move(x, y)

     cont(x, y)

     point(x, y)

     linemod(s)
     char s[];

     space(x0, y0, x1, y1)

     closepl()
```

DESCRIPTION
     These subroutines generate graphic output in a relatively
     device-independent manner. See plot(5) for a description of
     their effect. Openpl must be used before any of the others
     to open the device for writing. Closepl flushes the output.

     String arguments to label and linemod are null-terminated,
     and do not contain newlines.

     Various flavors of these functions exist for different out-
     put devices. They are obtained by the following ld(1)
     options:

```
     -lplot  device-independent graphics stream on standard out-
           put for plot(1) filters
     -l300   GSI 300 terminal
     -l300s  GSI 300S terminal
     -l450   GSI 450 terminal
     -l4013  Tektronix 4013 terminal
     -l4014  Tektronix 4014 and 4015 terminals with the Enhanced
           Graphics Module (Use -l4013 for 4014's or 4015's
           without the Enhanced Graphics Module)
     -lplotaed
           AED 512 color graphics terminal
     -lplotbg
           BBN bitgraph graphics terminal
     -lplotdumb
           Dumb terminals without cursor addressing or line
           printers
     -lplot  DEC Gigi terminals
     -lvt0   DEC vt100 terminals
     -lplot2648
           Hewlett Packard 2648 graphics terminal
     -lplot7221
           Hewlett Packard 7221 graphics terminal
     -lplotimagen
           Imagen laser printer (default 240 dots-per-inch
           resolution).
```

On many devices, it is necessary to pause after erase(), otherwise plotting commands are lost. The pause is normally done by the tty driver if at login time, tset found a df field in the termcap(5) entry for the terminal. If a pause is needed but not automatically being generated, add

```
   flush(stdout);
   sleep(1);
```

after each erase().

SEE ALSO
     plot(5), plot(1G), plot(3F), graph(1G)

NAME

popen, pclose - initiate I/O to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

DESCRIPTION

The arguments to popen are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO

pipe(2), fopen(3S), fclose(3S), system(3), wait(2), sh(1)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the shell cannot be accessed.

Pclose returns -1 if stream is not associated with a `popened' command.

BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, for instance, with fflush, see fclose(3S).

Popen always calls sh, never calls csh.

NAME
printf, fprintf, sprintf, vfprintf, vsprintf - formatted output conversion

SYNOPSIS
```
     #include <stdio.h>

     char *printf(format [, arg ] ...  )
     char *format;

     char *fprintf(stream, format [, arg ] ...      )
     FILE *stream;
     char *format;

     int sprintf(s, format [, arg ] ...  )
     char *s, *format;

     #include <varargs.h>
     char *vprintf(format, args)
     char *format;
     va_list args;

     char *vfprintf(stream, format, args)
     FILE *stream;
     char *format;
     va_list args;

     int vsprintf(s, format, args)
     char *s, *format;
     va_list args;
```

DESCRIPTION
Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places `output' in the string s, followed by the character `\0'. Alternate forms, in which the arguments have already been captured using the variable-length argument facilities of varargs(3), are available under the names vprintf, vfprintf, and vsprintf.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. The remainder of the conversion specification includes in the following order

+ a minus sign `-' which specifies left adjustment of the converted value in the indicated field;

+ an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;

+ an optional period, followed by an optional digit string giving a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

+ the character l specifying that a following d, o, x, or u corresponds to a long integer arg;

+ a character which indicates the type of conversion to be applied.

A field width or precision may be `*' instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

dox The integer arg is converted to signed decimal, unsigned octal, or unsigned hexadecimal notation respectively.

f The float or double arg is converted to decimal notation in the style `[-]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e The float or double arg is converted in the style `[-]d.ddde+dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

g The float or double arg is printed in style d, in style f, or in style e, whichever gives full precision in minimum space.

c The character arg is printed.

s   Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

u   The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 through MAX-UINT, where MAXUINT equals 4294967295 on a VAX-11 and 65535 on a PDP-11).

%   Print a `%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed as by putc(3S).

RETURN VALUE

The functions all return the number of characters printed, or -1 if an error occurred.

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, %02d:%02d", weekday, month, day,
     hour, min);
```

To print pi to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

putc(3S), scanf(3S)

BUGS

Very wide fields (>300 characters) fail.

Only sprintf and vsprintf return a count of characters transferred.

The functions still supports %D, %O, %U and %X. Do not use these formats, as they will be disappearing real soon now.

NAME
     psignal, sys_siglist - system signal messages

SYNOPSIS
```
     psignal(sig, s)
     unsigned sig;
     char *s;

     char *sys_siglist[];
```

DESCRIPTION
     Psignal produces a short message on the standard error file describing the indicated signal. First the argument string s is printed, then a colon, then the name of the signal and a new-line. Most usefully, the argument string is the name of the program which incurred the signal.   The signal number should be from among those found in <signal.h>.

     To simplify variant formatting of signal names, the vector of message strings sys_siglist is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define NSIG defined in <signal.h> is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO
     sigvec(2), perror(3)

NAME

putc, putchar, fputc, putw - put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

int putchar(c)

int fputc(c, stream)
FILE *stream;

int putw(w, stream)
FILE *stream;
```

DESCRIPTION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as putc(c, stdout).

Fputc behaves like putc, but is a genuine function rather than a macro.

Putw appends word (that is, int) w to the output stream. It returns the word written.   Putw neither assumes nor causes special alignment in the file.

SEE ALSO

fopen(3S), fclose(3S), getc(3S), puts(3S), printf(3S), fread(3S)

DIAGNOSTICS

These functions return the constant EOF upon error. Since this is a good integer, ferror(3S) should be used to detect putw errors.

BUGS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular

```
putc(c, *f++);
```

doesn't work sensibly.

Errors can occur long after the call to putc.

NAME

     puts, fputs - put a string on a stream

SYNOPSIS

```
     #include <stdio.h>

     puts(s)
     char *s;

     fputs(s, stream)
     char *s;
     FILE *stream;
```

DESCRIPTION

     Puts copies the null-terminated string s to the standard
     output stream stdout and appends a newline character.

     Fputs copies the null-terminated string s to the named out-
     put stream.

     Neither routine copies the terminal null character.

SEE ALSO

     fopen(3S), gets(3S), putc(3S), printf(3S), ferror(3S)
     fread(3S) for fwrite

BUGS

     Puts appends a newline, fputs does not, all in the name of
     backward compatibility.

NAME
     qsort - quicker sort

SYNOPSIS
```
     qsort(base, nel, width, compar)
     char *base;
     int (*compar)();
```

DESCRIPTION
     Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO
     sort(1)

NAME

rand, srand - random number generator

SYNOPSIS

```
srand(seed)
int seed;

rand()
```

DESCRIPTION

The newer random(3) should be used in new applications; rand remains for compatibilty.

Rand uses a multiplicative congruential random number generator with period 2**32 to return successive pseudo-random numbers in the range from 0 to (2**31)-1 on the VAX, and (2**15)-1 on the PDP-11.

The generator is reinitialized by calling srand with 1 as argument.   It can be set to a random starting point by calling srand with whatever you like as argument.

SEE ALSO

random(3)

NAME

random, srandom, initstate, setstate - better random number generator; routines for changing generators

SYNOPSIS

```
long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;
```

DESCRIPTION

Random uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to (2**31)-1. The period of this random number generator is very large, approximately 16*((2**31)-1).

Random/srandom have (almost) the same calling sequence and initialization properties as rand/srand. The difference is that rand(3) produces a much less random sequence - in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by random are usable. For example, ``random()&01'' will produce a random binary value.

Unlike srand, srandom does not return the old seed; the reason for this is that the amount of state information used is much more than a single word. (Two other routines are provided to deal with restarting/changing random number generators). Like rand(3), however, random will by default produce a sequence of numbers that can be duplicated by calling srandom with 1 as the seed.

The initstate routine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by initstate to decide how sophisticated a random number generator it should use -- the more state, the better the random numbers will be. (Current "optimal" values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error). The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. Initstate returns a pointer to the previous state information array.

Once a state has been initialized, the setstate routine provides for rapid switching between states.   Setstate returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to initstate or setstate.

Once a state array has been initialized, it may be restarted at a different point either by calling initstate (with the desired seed, the state array, and its size) or by calling both setstate (with the state array) and srandom (with the desired seed). The advantage of calling both setstate and srandom is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than 2**69 which should be sufficient for most purposes.

AUTHOR

Earl T. Cohen

DIAGNOSTICS

If initstate is called with less than 8 bytes of state information, or if setstate detects that the state information has been garbled, error messages are printed on the standard error output.

SEE ALSO

rand(3)

BUGS

About 2/3 the speed of rand(3C).

NAME

rcmd, rresvport, ruserok - routines for returning a stream to a remote command

SYNOPSIS

```
rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
int inport;
char *locuser, *remuser, *cmd;
int *fd2p;

s = rresvport(port);
int *port;

ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

Rcmd is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. Rresvport is a routine which returns a descriptor to a socket with an address in the privileged port space. Ruserok is a routine used by servers to authenticate clients requesting service with rcmd. All three functions are present in the same file and are used by the rshd(8C) server (among others).

Rcmd looks up the host *ahost using gethostbyname(3N), returning -1 if the host does not exist. Otherwise *ahost is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port inport.

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If fd2p is nonzero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in *fd2p. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. If fd2p is 0, then the stderr (unit 2 of the remote command) will be made the same as the stdout and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in rshd(8C).

The rresvport routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by rcmd and several other routines. Privileged Internet ports are those in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

Ruserok takes a remote host's name, as returned by a gethostbyaddr(3N) routine, two user names and a flag indicating whether the local user's name is that of the super-user. It then checks the files /etc/hosts.equiv and, possibly, .rhosts in the current working directory (normally the local user's home directory) to see if the request for service is allowed. A 0 is returned if the machine name is listed in the ``hosts.equiv'' file, or the host and remote user name are found in the ``.rhosts'' file; otherwise ruserok returns -1. If the superuser flag is 1, the checking of the ``host.equiv'' file is bypassed. If the local domain (as obtained from gethostname(2)) is the same as the remote domain, only the machine name need be specified.

SEE ALSO

rlogin(1C), rsh(1C), intro(2), rexec(3), rexecd(8C), rlogind(8C), rshd(8C)

DIAGNOSTICS

Rcmd returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error.

Rresvport returns a valid, bound socket descriptor on success. It returns -1 on error with the global value errno set according to the reason for failure. The error code EAGAIN is overloaded to mean ``All network ports in use.''

NAME

re_comp, re_exec - regular expression handler

SYNOPSIS

```
char *re_comp(s)
char *s;

re_exec(s)
char *s;
```

DESCRIPTION

Re_comp compiles a string into an internal form suitable for pattern matching. Re_exec checks the argument string against the last string passed to re_comp.

Re_comp returns 0 if the string s was compiled successfully; otherwise a string containing an error message is returned. If re_comp is passed 0 or a null string, it returns without changing the currently compiled regular expression.

Re_exec returns 1 if the string s matches the last compiled regular expression, 0 if the string s failed to match the last compiled regular expression, and -1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both re_comp and re_exec may have trailing or embedded newline characters; they are terminated by nulls.   The regular expressions recognized are described in the manual entry for ed(1), given the above difference.

SEE ALSO

ed(1), ex(1), egrep(1), fgrep(1), grep(1)

DIAGNOSTICS

Re_exec returns -1 for an internal error.

Re_comp returns one of the following strings if an error occurs:

```
No previous regular expression,
Regular expression too long,
unmatched \(,
missing ],
too many \(\) pairs,
unmatched \).
```

NAME

res_mkquery, res_send, res_init, dn_comp, dn_expand - resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr,
buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, eomorig, comp_dn, exp_dn, length)
char *msg, *eomorig, *comp_dn, exp_dn;
int length;
```

DESCRIPTION

These routines are used for making, sending and interpreting packets for use with Internet domain name servers. Global information that is used by the resolver routines is kept in the variable _res. Most of the values have reasonable defaults and can be ignored. Options stored in _res.options are defined in resolv.h and are as follows. Options are stored a simple bit mask containing the bitwise ``or'' of the options enabled.

RES_INIT
  True if the initial name server address and default domain name are initialized (i.e., res_init has been called).

RES_DEBUG
   Print debugging messages.

RES_AAONLY
   Accept authoritative answers only. With this option, res_send should continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
   Use TCP connections for queries instead of UDP datagrams.

RES_STAYOPEN
   Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC
   Unused currently (ignore truncation errors, i.e., don't retry with TCP).

RES_RECURSE
   Set the recursion-desired bit in queries. This is the default. (res_send does not do iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
   If set, res_mkquery will append the default domain name to single-component names (those that do not contain a dot). This is the default.

RES_DNSRCH
   If this option is set, the standard host lookup routine gethostbyname(3) will search for host names in the current domain and in parent domains; see hostname(7).

Res_init

reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. Res_mkquery makes a standard query message and places it in buf. Res_mkquery will return the size of the query or -1 if the query is larger than buflen. Op is usually QUERY but can be any of the query types defined in nameser.h. Dname is the domain name. If dname consists of a single label and the RES_DEFNAMES flag is enabled (the default), the current domain name will be appended to dname. The current domain name is defined by the hostname or is specified in a system file; it can be overridden by the environment variable LOCALDOMAIN. Newrr is currently unused but is intended for making update messages.

Res_send sends a query to name servers and returns an answer. It will call res_init if RES_INIT is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned, or -1 if there were errors.

Dn_expand expands the compressed domain name comp_dn to a full domain name. Expanded names are converted to upper case. Msg is a pointer to the beginning of the message, exp_dn is a pointer to a buffer of size length for the result. The size of compressed name is returned or -1 if there was an error.

Dn_comp compresses the domain name exp_dn and stores it in comp_dn. The size of the compressed name is returned or -1 if there were errors. length is the size of the comp_dn. Dnptrs is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message and the list ends with NULL. lastdnptr is a pointer to the end of the array pointed to dnptrs. A side effect is to update the list of pointers for labels inserted into the message by dn_comp as the name is compressed. If dnptr is NULL, names are not compressed. If lastdnptr is NULL, the list of labels is not updated.

FILES

/etc/resolv.conf   see resolver(5)

SEE ALSO

gethostbyname(3), named(8), resolver(5), hostname(7),
RFC882, RFC883, RFC973, RFC974,
SMM:11 Name Server Operations Guide for BIND

NAME
rexec - return stream to a remote command

SYNOPSIS
```
     rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
     char **ahost;
     int inport;
     char *user, *passwd, *cmd;
     int *fd2p;
```

DESCRIPTION
Rexec looks up the host *ahost using gethostbyname(3N), returning -1 if the host does not exist. Otherwise *ahost is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's .netrc file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port inport specifies which well-known DARPA Internet port to use for the connection; the call ``getservbyname("exec", "tcp")'' (see getservent(3N)) will return a pointer to a structure, which contains the necessary port. The protocol for connection is described in detail in rexecd(8C).

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If fd2p is non-zero, then an auxiliary channel to a control process will be setup, and a descriptor for it will be placed in *fd2p. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. The diagnostic information returned does not include remote authorization failure, as the secondary connection is set up after authorization has been verified. If fd2p is 0, then the stderr (unit 2 of the remote command) will be made the same as the stdout and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO
rcmd(3), rexecd(8C)

NAME

scandir, alphasort - scan a directory

SYNOPSIS

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

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct *(*namelist[]);
int (*select)();
int (*compar)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

DESCRIPTION

Scandir reads the directory dirname and builds an array of pointers to directory entries using malloc(3). It returns the number of entries in the array and a pointer to the array through namelist.

The select parameter is a pointer to a user supplied subroutine which is called by scandir to select which entries are to be included in the array. The select routine is passed a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If select is null, then all the directory entries will be included.

The compar parameter is a pointer to a user supplied subroutine which is passed to qsort(3) to sort the completed array. If this pointer is null, the array is not sorted. Alphasort is a routine which can be used for the compar parameter to sort the array alphabetically.

The memory allocated for the array can be deallocated with free (see malloc(3)) by freeing each pointer in the array and the array itself.

SEE ALSO

directory(3), malloc(3), qsort(3), dir(5)

DIAGNOSTICS

Returns -1 if the directory cannot be opened for reading or if malloc(3) cannot allocate enough memory to hold all the data structures.

NAME
scanf, fscanf, sscanf - formatted input conversion

SYNOPSIS
```
#include <stdio.h>

scanf(format [ , pointer ] . . .  )
char *format;

fscanf(stream, format [ , pointer ] . . .     )
FILE *stream;
char *format;

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.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.

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, and a conversion character.

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 *. 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 character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

%   a single `%' is expected in the input at this point; no assignment is done.

d   a decimal integer is expected; the corresponding argument should be an integer pointer.

o   an octal integer is expected; the corresponding argument should be a integer pointer.

x   a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s   a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating `\0', which will be added. The input field is terminated by a space character or a newline.

c   a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try `%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

f

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 e followed by an optionally signed integer.

[indicates a string not to be delimited by space characters.   The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o and x may be capitalized or preceded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e or f may be capitalized or preceded by l to indicate a pointer to double rather than to float. The conversion characters d, o and x may be preceded by h to indicate a pointer to short rather than to int.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
        int i; float x; char name[50];
        scanf("%d%f%s", &i, &x, name);
```

with the input line

```
   25   54.32E-1  thompson
```

will assign to i the value 25, x the value 5.432, and name will contain `thompson\0'. Or,

```
   int i; float x; char name[50];
   scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
   56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip `0123', and place the string `56\0' in name. The next call to getchar will return `a'.

SEE ALSO

atof(3), getc(3S), printf(3S)

DIAGNOSTICS

The scanf functions return EOF on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

setbuf, setbuffer, setlinebuf, setvbuf -stream buffering operations

SYNOPSIS

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

void
setbuf(stream, buf)
     FILE *stream;
     char *buf;

void
setbuffer(stream, buf, size)
     FILE *stream;
     char *buf;
     size_t size;

int
setlinebuf(stream)
     FILE *stream;

int
setvbuf(stream, buf, mode, size)
     FILE *stream;
     char *buf;
     int mode;
     size_t size
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is output or input is read from any stream attached to a terminal device (typically stdin). The function fflush(3) may be used to force the block out early. (See fclose(3).)

Normally all files are block buffered. When the first I/O operation occurs on a file, malloc(3) is called, and an optimally-sized buffer is obtained. If a stream refers to a terminal (as stdout normally does) it is line buffered. The standard error stream stderr is always unbuffered.

The setvbuf function may be used to alter the buffering behavior of a stream. The mode parameter must be one of the following three macros:

```
   _IONBF    unbuffered

   _IOLBF    line buffered

   _IOFBF    fully buffered
```

The size parameter may be given as zero to obtain deferred optimal-size buffer allocation as usual. If it is not zero, then except for unbuffered files, the buf argument should point to a buffer at least size bytes long; this buffer will be used instead of the current buffer. (If the size argument is not zero but buf is NULL, a buffer of the given size will be allocated immediately, and released on close. This is an extension to ANSI C; portable code should use a size of 0 with any NULL buffer.)

The setvbuf function may be used at any time, but may have peculiar side effects (such as discarding input or flushing output) if the stream is ``active''. Portable applications should call it only once on any given stream, and before any I/O is performed.

The other three calls are, in effect, simply aliases for calls to setvbuf. Except for the lack of a return value, the setbuf function is exactly equivalent to the call

```
   setvbuf(stream, buf, buf ? _IOFBF : _IONBF, BUFSIZ);
```

The setbuffer function is the same, except that the size of the buffer is up to the caller, rather than being determined by the default BUFSIZ. The setlinebuf function is exactly equivalent to the call:

```
   setvbuf(stream, (char *)NULL, _IOLBF, 0);
```

RETURN VALUES

The setvbuf function returns 0 on success, or EOF if the request cannot be honored (note that the stream is still functional in this case).

The setlinebuf function returns what the equivalent setvbuf would have returned.

SEE ALSO

fopen(3), fclose(3), fread(3), malloc(3), puts(3), printf(3)

STANDARDS

The setbuf and setvbuf functions conform to ANSI C X3.159-1989 (``ANSI C'').

BUGS

The setbuffer and setlinebuf functions are not portable to

versions of BSD before 4.2BSD. On 2BSD systems, setbuf
always uses a 1kb buffer size.

NAME

setjmp, longjmp - non-local goto

SYNOPSIS

```
#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;

_setjmp(env)
jmp_buf env;

_longjmp(env, val)
jmp_buf env;
```

DESCRIPTION

These routines 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. It returns value 0.

Longjmp restores the environment saved by the last call of setjmp. It then returns in such a way that execution continues as if the call of setjmp had just returned the value val to the function that invoked setjmp, which must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

Setjmp and longjmp save and restore the signal mask sigmask(2), while _setjmp and _longjmp manipulate only the C stack and registers.

ERRORS

If the contents of the jmp_buf are corrupted, or correspond to an environment that has already returned, longjmp calls the routine longjmperror.   If longjmperror returns the program is aborted. The default version of longjmperror prints the message ``longjmp botch'' to standard error and returns. User programs wishing to exit more gracefully can write their own versions of longjmperror.

SEE ALSO

sigvec(2), sigstack(2), signal(3)

NOTES (PDP-11)

On the PDP-11, longjmperror is called as _ljerr. This difference stems from the limited name size of the PDP-11 that requires all external names to be unique within the first seven characters. However, <setjmp.h> automatically translates longjmperror to ljerror and should be included before any definition longjmperror.

The PDP-11 implementation also contains a subtle bug that occurs when a routine containing a setjmp has register variables. The bug sometimes causes those variables to be given invalid values when a longjmp is made back to the routine. Register variables should therefore be avoided in routines containing setjmps.

And finally, _longjmp may sometimes die fatally. Sorry.

NAME

getmode, setmode - modify mode bits

SYNOPSIS

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

mode_t
getmode(set, mode)
void *set;
mode_t mode

void *
setmode(mode_str)
char *mode_str;
```

DESCRIPTION

The getmode function returns a copy of the file permission bits mode as altered by the values pointed to by set. While only the mode bits are altered, other parts of the file mode may be examined.

The setmode function takes an absolute (octal) or symbolic value, as described in chmod(1), as an argument and returns a pointer to mode values to be supplied to getmode. Because some of the symbolic values are relative to the file creation mask, setmode may call umask(2). If this occurs, the file creation mask will be restored before setmode returns. If the calling program changes the value of its file creation mask after calling setmode, setmode must be called again if getmode is to modify future file modes correctly.

If the mode passed to setmode is invalid, setmode returns NULL.

ERRORS

The setmode function may fail and set errno for any of the errors specified for the library routine malloc(3).

SEE ALSO

chmod(1), stat(2), umask(2), malloc(3)

HISTORY

The getmode and setmode functions first appeared in 4.4BSD.

NAME
     setruid, setrgid - set user and group ID

SYNOPSIS
```
     #include <sys/types.h>
     int
     setruid(ruid)
       uid_t ruid

     int
     setrgid(rgid)
       gid_t rgid
```

DESCRIPTION
     The setruid function (setrgid) sets the real user ID (group
     ID) of the current process.

RETURN VALUES
     Upon success, these functions return 0; otherwise -1 is
     returned.

     If the user is not the super user, or the uid specified is
     not the real or effective ID, these functions return -1.

     The use of these calls is not portable. Their use is
     discouraged; they will be removed in the future.

SEE ALSO
     setuid(2), setgid(2), seteuid(2), setegid(2), getuid(2),
     getgid(2)

HISTORY
     The setruid and setrgid syscalls appeared in 4.2BSD and were
     dropped in 4.4BSD.

NAME

siginterrupt - allow signals to interrupt system calls

SYNOPSIS

```
siginterrupt(sig, flag);
int sig, flag;
```

DESCRIPTION

Siginterrupt is used to change the system call restart behavior when a system call is interrupted by the specified signal. If the flag is false (0), then system calls will be restarted if they are interrupted by the specified signal and no data has been transferred yet. System call restart is the default behavior on 4.2 BSD.

If the flag is true (1), then restarting of system calls is disabled. If a system call is interrupted by the specified signal and no data has been transferred, the system call will return -1 with errno set to EINTR. Interrupted system calls that have started transferring data will return the amount of data actually transferred. System call interrupt is the signal behavior found on 4.1 BSD and AT&T System V UNIX systems.

Note that the new 4.2 BSD signal handling semantics are not altered in any other way. Most notably, signal handlers always remain installed until explicitly changed by a subsequent sigvec(2) call, and the signal mask operates as documented in sigvec(2). Programs may switch between restartable and interruptible system call operation as often as desired in the execution of a program.

Issuing a siginterrupt(3) call during the execution of a signal handler will cause the new action to take place on the next signal to be caught.

NOTES

This library routine uses an extension of the sigvec(2) system call that is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUE

A 0 value indicates that the call succeeded. A -1 value indicates that an invalid signal number has been supplied.

SEE ALSO

sigvec(2), sigblock(2), sigpause(2), sigsetmask(2).

NAME

signal - simplified software signal facilities

SYNOPSIS

```
#include <signal.h>

(*signal(sig, func))()
int (*func)();
```

DESCRIPTION

Signal is a simplified interface to the more general sigvec(2) facility.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see tty(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the signal call allows signals either to be ignored or to cause an interrupt to a specified location. The following is a list of all signals with names as in the include file <signal.h>:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	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
SIGURG	16@	urgent condition present on socket
SIGSTOP	17'+'	stop (cannot be caught or ignored)
SIGTSTP	18'+'	stop signal generated from keyboard
SIGCONT	19@	continue after stop
SIGCHLD	20@	child status has changed
SIGTTIN	21'+'	background read attempted from control terminal
SIGTTOU	22'+'	background write attempted to control terminal

```
SIGIO     23@  i/o is possible on a descriptor (see fcntl(2))
SIGXCPU   24   cpu time limit exceeded (see setrlimit(2))
SIGXFSZ   25   file size limit exceeded (see setrlimit(2))
SIGVTALRM 26   virtual time alarm (see setitimer(2))
SIGPROF   27   profiling timer alarm (see setitimer(2))
SIGWINCH  28@  Window size change
SIGUSR1   30   User defined signal 1
SIGUSR2   31   User defined signal 2
```

The starred signals in the list above cause a core image if not caught or ignored.

If func is SIG_DFL, the default action for signal sig is reinstated; this default is termination (with a core image for starred signals) except for signals marked with @ or '+'. Signals marked with @ are discarded if the action is SIG_DFL; signals marked with '+' cause the process to stop. If func is SIG_IGN the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and func is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. Unlike previous signal facilities, the handler func remains installed after a signal has been delivered.

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is automatically restarted. In particular this can occur during a read or write(2) on a slow device (such as a terminal; but not a file) and during a wait(2).

The value of signal is the previous (or initial) value of func for the particular signal.

After a fork(2) or vfork(2) the child inherits all signals. Execve(2) resets all caught signals to the default action; ignored signals remain ignored.

RETURN VALUE

The previous action is returned on a successful call. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS

Signal will fail and no action will take place if one of the following occur:

[EINVAL]	Sig is not a valid signal number.
[EINVAL]	An attempt is made to ignore or supply a

handler for SIGKILL or SIGSTOP.

[EINVAL] An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

SEE ALSO

kill(1), ptrace(2), kill(2), sigvec(2), sigblock(2), sigsetmask(2), sigpause(2), sigstack(2), setjmp(3), tty(4)

NOTES (VAX-11)

The handler routine can be declared:

```
  handler(sig, code, scp)
```

Here sig is the signal number, into which the hardware faults and traps are mapped as defined below. Code is a parameter which is either a constant as given below or, for compatibility mode faults, the code provided by the hardware. Scp is a pointer to the struct sigcontext used by the system to restore the process context from before the signal. Compatibility mode faults are distinguished from the other SIGILL traps by having PSL_CM set in the psl.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in <signal.h>:

Hardware condition	Signal	Code
Arithmetic traps:		
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Subscript-range	SIGFPE	FPE_SUBRNG_TRAP
Floating overflow fault	SIGFPE	FPE_FLTOVF_FAULT
Floating divide by zero fault	SIGFPE	FPE_FLTDIV_FAULT
Floating underflow fault	SIGFPE	FPE_FLTUND_FAULT
Length access control	SIGSEGV	
Protection violation	SIGBUS	
Reserved instruction	SIGILL	ILL_RESAD_FAULT
Customer-reserved instr.	SIGEMT	
Reserved operand	SIGILL	ILL_PRIVIN_FAULT
Reserved addressing	SIGILL	ILL_RESOP_FAULT
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	
Compatibility-mode	SIGILL	hardware supplied code
Chme	SIGSEGV	
Chms	SIGSEGV	
Chmu	SIGSEGV	

NOTES (PDP-11)

The handler routine can be declared:

```
handler(sig, code, scp)
int sig, code;
struct sigcontext *scp;
```

Here sig is the signal number, into which the hardware faults and traps are mapped as defined below. Code is a parameter that is a constant as given below. Scp is a pointer to the sigcontext structure (defined in <signal.h>), used to restore the context from before the signal.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in <signal.h>:

Hardware condition	Signal	Code
Arithmetic traps:		
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Illegal return code	SIGFPE	FPE_CRAZY
Bad op code	SIGFPE	FPE_OPCODE_TRAP
Bad operand	SIGFPE	FPE_OPERAND_TRAP
Maintenance trap	SIGFPE	FPE_MAINT_TRAP
Length access control	SIGSEGV	
Protection violation (odd address)	SIGBUS	
Reserved instruction	SIGILL	ILL_RESAD_FAULT
Customer-reserved instr.	SIGEMT	
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	

The handler routine must save any registers it uses and restore them before returning. On the PDP-11, the kernel saves r0 and r1 before calling the handler routine, but expect the handler to save any other registers it uses. The standard entry code generated by the C compiler for handler routines written in C automatically saves the remaining general registers, but floating point registers are not saved. As a result there is currently no [standard] method for a handler routine written in C to perform floating point operations without blowing the interrupted program out of the water.

```
NAME
     sigemptyset, sigfillset, sigaddset, sigdelset, sigismember -
     manipulate signal sets

SYNOPSIS
     #include <signal.h>

     sigemptyset(set)
     sigset_t *set;

     sigfillset(set)
     sigset_t *set;

     sigaddset(set, signo)
     sigset_t *set;
     int signo;

     sigdelset(set, signo)
     sigset_t *set;
     int signo;

     sigismember(set, signo)
     sigset_t *set;
     int signo;

DESCRIPTION
     These functions manipulate signal sets stored in a sigset_t
     . Either sigemptyset or sigfillset must be called for every
     object of type sigset_t before any other use of the object.

     The sigemptyset function initializes a signal set to be
     empty.

     The sigfillset function initializes a signal set to contain
     all signals.

     The sigaddset function adds the specified signal signo to
     the signal set.

     The sigdelset function deletes the specified signal signo
     from the signal set.

     The sigismember function returns whether a specified signal
     signo is contained in the signal set.

     These functions are provided as macros in the include file
     <signal.h>.  Actual functions are available if their names
     are undefined (with #undef name).

RETURN VALUES
     The sigismember function returns 1 if the signal is a member
     of the set, 0 otherwise.  The other functions return 0.
```

ERRORS

Currently no errors are detected.

SEE ALSO

kill(2), sigaction(2), sigsuspend(2)

STANDARDS

These functions are defined by IEEE Std1003.1-1988 (``POSIX'').

NAME
     sin, cos, tan, asin, acos, atan, atan2 - trigonometric functions and their inverses

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 y,x;
```

DESCRIPTION
     Sin, cos and tan return trigonometric functions of radian arguments x.

     Asin returns the arc sine in the range -pi/2 to pi/2.

     Acos returns the arc cosine in the range 0 to pi.

     Atan returns the arc tangent in the range -pi/2 to pi/2.

     On a VAX,
```
     atan2(y,x) :=   atan(y/x)                 if x > 0,
                sign(y)*(pi - atan(|y/x|))  if x < 0,
                0                         if x = y = 0, or
                sign(y)*pi/2       if x = 0 != y.
```

DIAGNOSTICS
     On a VAX, if |x| > 1 then asin(x) and acos(x) will return reserved operands and errno will be set to EDOM.

NOTES
     Atan2 defines atan2(0,0) = 0 on a VAX despite that previously atan2(0,0) may have generated an error message. The reasons for assigning a value to atan2(0,0) are these:

(1) Programs that test arguments to avoid computing atan2(0,0) must be indifferent to its value. Programs that require it to be invalid are vulnerable to diverse reactions to that invalidity on diverse computer systems.

(2) Atan2 is used mostly to convert from rectangular (x,y) to polar (r,theta) coordinates that must satisfy x = r*cos theta and y = r*sin theta. These equations are satisfied when (x=0,y=0) is mapped to (r=0,theta=0) on a VAX. In general, conversions to polar coordinates should be computed thus:

```
        r := hypot(x,y);     ... := sqrt(x*x+y*y)
    theta := atan2(y,x).
```

(3) The foregoing formulas need not be altered to cope in a reasonable way with signed zeros and infinities on a machine that conforms to IEEE 754; the versions of hypot and atan2 provided for such a machine are designed to handle all cases. That is why atan2(+0,-0) = +pi, for instance. In general the formulas above are equivalent to these:

```
r := sqrt(x*x+y*y); if r = 0 then x := copysign(1,x);
if x > 0  then theta := 2*atan(y/(r+x))
        else theta := 2*atan((r-x)/y);
```

except if r is infinite then atan2 will yield an appropriate multiple of pi/4 that would otherwise have to be obtained by taking limits.

ERROR (due to Roundoff etc.)

Let P stand for the number stored in the computer in place of pi = 3.14159 26535 89793 23846 26433 ... . Let "trig" stand for one of "sin", "cos" or "tan". Then the expression "trig(x)" in a program actually produces an approximation to trig(x*pi/P), and "atrig(x)" approximates (P/pi)*atrig(x). The approximations are close, within 0.9 ulps for sin, cos and atan, within 2.2 ulps for tan, asin, acos and atan2 on a VAX. Moreover, P = pi in the codes that run on a VAX.

In the codes that run on other machines, P differs from pi by a fraction of an ulp; the difference matters only if the argument x is huge, and even then the difference is likely to be swamped by the uncertainty in x. Besides, every trigonometric identity that does not involve pi explicitly is satisfied equally well regardless of whether P = pi. For instance, sin(x)**2+cos(x)**2 = 1 and sin(2x) = 2sin(x)cos(x) to within a few ulps no matter how big x may be. Therefore the difference between P and pi is most unlikely to affect scientific and engineering computations.

SEE ALSO

math(3M), hypot(3M), sqrt(3M), infnan(3M)

AUTHOR

Robert P. Corbett, W. Kahan, Stuart I. McDonald, Peter Tang and, for the codes for IEEE 754, Dr. Kwok-Choi Ng.

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
     These functions compute the designated hyperbolic functions for real arguments.

ERROR (due to Roundoff etc.)
     Below 2.4 ulps; an ulp is one Unit in the Last Place.

DIAGNOSTICS
     Sinh and cosh return the reserved operand on a VAX if the correct value would overflow.

SEE ALSO
     math(3M), infnan(3M)

AUTHOR
     W. Kahan, Kwok-Choi Ng

NAME

sleep, usleep - suspend process execution

SYNOPSIS

```
#include <unistd.h>

unsigned int
sleep(seconds)
   unsigned int seconds;

void
usleep(microseconds)
    long microseconds;
```

DESCRIPTION

The sleep function suspends execution of the calling process for seconds of clock time, or until interrupted by a signal.

The usleep function suspends execution of the calling process for microseconds of clock time, or until interrupted by a signal.

System activity may lengthen the suspension.

RETURN VALUES

The sleep function returns 0, or if interrupted before seconds, the amount not slept (the requested time minus the time actually slept) in seconds. The usleep function does not return anything (meaningful).

SEE ALSO

select(2)

COMPATIBILITY

Previous implementations of sleep and usleep re-suspended the process if interrupted by a signal. This implementation has been changed to return in that case, to conform to POSIX 1003.1-88.

On the PDP-11 the previous version of usleep took a u_int as the input parameter. This has been changed to be long so that usleep can be used for more than 65 milliseconds (a u_int could only count 65535 microseconds) of sleep. Thus it is now possible for usleep to handle longer sleep durations than sleep.

BUGS

On the PDP-11 the clock resolution is limited to the line frequency (usually 60Hz in the U.S.A. and 50Hz elsewhere).

HISTORY

A usleep function appeared in 4.3BSD. A sleep function appeared in V7.

```
NAME
     cbrt, sqrt - cube root, square root

SYNOPSIS
     #include <math.h>

     double cbrt(x)
     double x;

     double sqrt(x)
     double x;

DESCRIPTION
     Cbrt(x) returns the cube root of x.

     Sqrt(x) returns the square root of x.

DIAGNOSTICS
     On a VAX, sqrt(negative) returns the reserved operand and
     sets errno to EDOM .

ERROR (due to Roundoff etc.)
     Cbrt is accurate to within 0.7 ulps.
     Sqrt on a VAX is accurate to within 0.501 ulps.
     Sqrt on a machine that conforms to IEEE 754 is correctly
     rounded in accordance with the rounding mode in force; the
     error is less than half an ulp in the default mode
     (round-to-nearest).  An ulp is one Unit in the Last Place
     carried.

SEE ALSO
     math(3M), infnan(3M)

AUTHOR
     W. Kahan
```

NAME

stdio - standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The functions described in section 3S constitute a user-level buffering scheme. The in-line macros getc and putc(3S) handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite 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. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant `pointer' NULL (0) designates no stream at all.

An integer constant EOF (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following `functions' are implemented as macros; redeclaration of these names is perilous: getc, getchar, putc, putchar, feof, ferror, fileno.

SEE ALSO

open(2), close(2), read(2), write(2), fread(3S), fseek(3S), f*(3S)

DIAGNOSTICS

The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized with fopen, input (output) has been attempted on an output (input) stream, or a FILE pointer designates corrupt or otherwise unintelligible FILE data.

For purposes of efficiency, this implementation of the standard library has been changed to line buffer output to a terminal by default and attempts to do this transparently by flushing the output whenever a read(2) from the standard input is necessary. This is almost always transparent, but may cause confusion or malfunctioning of programs which use standard i/o routines but use read(2) themselves to read from the standard input.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to fflush(3S) the standard output before going off and computing so that the output will appear.

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially vfork and abort.

LIST OF FUNCTIONS

Name	Appears on Page	Description
Clearer	ferror.3s	stream status inquiries
fclose	fclose.3s	close or flush a stream
fdopen	fopen.3s	open a stream
feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fflush	fclose.3s	close or flush a stream
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
fopen	fopen.3s	open a stream
fprintf	printf.3s	formatted output conversion
fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream
fread	fread.3s	buffered binary input/output
freopen	fopen.3s	open a stream
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3s	reposition a stream
ftell	fseek.3s	reposition a stream
fwrite	fread.3s	buffered binary input/output
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
gets	gets.3s	get a string from a stream
getw	getc.3s	get character or word from stream
printf	printf.3s	formatted output conversion
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream

rewind	fseek.3s	reposition a stream
scanf	scanf.3s	formatted input conversion
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setlinebuf	setbuf.3s	assign buffering to a stream
sprintf	printf.3s	formatted output conversion
sscanf	scanf.3s	formatted input conversion
ungetc	ungetc.3s	push character back into input stream

NAME

strcspn - span the complement of a string

SYNOPSIS

```
#include <string.h>

size_t
strcspn(s, charset)
char *s;
char *charset;
```

DESCRIPTION

The strcspn() function spans the initial part of the null-terminated string s as long as the characters from s do not occur in string charset (it spans the complement of charset) .

RETURN VALUES

The strcspn() function returns the number of characters spanned.

SEE ALSO

index(3), memchr(3), rindex(3), strchr(3), strpbrk(3), strrchr(3), strsep(3), strspn(3), strstr(3), strtok(3)

STANDARDS

The strcspn() function conforms to ANSI C X3.159-1989 (``ANSI C'').

NAME
strftime - format date and time

SYNOPSIS
```
#include <sys/types.h>
#include <time.h>
#include <string.h>

size_t strftime(buf, maxsize, format, timeptr)
char *buf;
size_t maxsize;
char *format;
struct tm *timeptr;
```

DESCRIPTION
The strftime() function formats the information from timeptr into the buffer buf according to the string pointed to by format.

The format string consists of zero or more conversion specifications and ordinary characters. All ordinary characters are copied directly into the buffer. A conversion specification consists of a percent sign ``%'' and one other character.

No more than maxsize characters will be placed into the array. If the total number of resulting characters, including the terminating null character, is not more than maxsize, strftime() returns the number of characters in the array, not counting the terminating null. Otherwise, zero is returned.

Each conversion specification is replaced by the characters as follows which are then copied into the buffer.

%A   is replaced by the full weekday name.

%a   is replaced by the abbreviated weekday name, where the abbreviation is the first three characters.

%B   is replaced by the full month name.

%b or %h
is replaced by the abbreviated month name, where the abbreviation is the first three characters.

%C is equivalent to ``%a %b %e %H:%M:%S %Y'' (the format produced by asctime(3)).

%c   is equivalent to ``%m/%d/%y''.

%D   is replaced by the date in the format ``mm/dd/yy''.

```
     %d    is replaced by the day of the month as a decimal
          number (01-31).

     %e    is replaced by the day of month as a decimal number
          (1-31); single digits are preceded by a blank.

     %H    is replaced by the hour (24-hour clock) as a decimal
          number (00-23).

     %I    is replaced by the hour (12-hour clock) as a decimal
          number (01-12).

     %j    is replaced by the day of the year as a decimal
          number (001-366).

     %k    is replaced by the hour (24-hour clock) as a decimal
          number (0-23); single digits are preceded by a blank.

     %l    is replaced by the hour (12-hour clock) as a decimal
          number (1-12); single digits are preceded by a blank.

     %M    is replaced by the minute as a decimal number (00-
          59).

     %m    is replaced by the month as a decimal number (01-12).

     %n    is replaced by a newline.

     %p    is replaced by either ``AM'' or ``PM'' as appropri-
          ate.

     %R    is equivalent to ``%H:%M''

     %r    is equivalent to ``%I:%M:%S %p'' .

     %t    is replaced by a tab.

     %S    is replaced by the second as a decimal number (00-
          60).

     %T or %X
          is equivalent to "%H:%M:%S" .

     %U    is replaced by the week number of the year (Sunday as
          the first day of the week) as a decimal number (00-
          53).

     %W    is replaced by the week number of the year (Monday as
          the first day of the week) as a decimal number (00-
          53).

     %w    is replaced by the weekday (Sunday as the first day
          of the week) as a decimal number (0-6).

     %x     is equivalent to ``%m/%d/%y %H:%M:%S'' .

     %Y     is replaced by the year with century as a decimal
          number.

     %y     is replaced by the year without century as a decimal
          number (00-99).

     %Z     is replaced by the time zone name.

     %%     is replaced by `%' .
```

SEE ALSO
     date(1), ctime(3), printf(1), printf(3)

STANDARDS
     The strftime() function conforms to ANSI X C3.159-
     1989(``ANSI C'').

BUGS
     There is no conversion specification for the phase of the
     moon.

NAME

strcat, strncat, strcmp, strncmp, strcasecmp, strncasecmp, strcpy, strncpy, strlen, index, rindex - string operations

SYNOPSIS

```
#include <strings.h>

char *strcat(s, append)
char *s, *append;

char *strncat(s, append, count)
char *s, *append;
int count;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, count)
char *s1, *s2;
int count;

strcasecmp(s1, s2)
char *s1, *s2;

strncasecmp(s1, s2, count)
char *s1, *s2;
int count;

char *strcpy(to, from)
char *to, *from;

char *strncpy(to, from, count)
char *to, *from;
int count;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string append to the end of string s. Strncat copies at most count characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2. Strncmp makes the same comparison but looks at at most count characters. Strcasecmp and strncasecmp are identical in function, but are case insensitive. The returned lexicographic difference reflects a conversion to lower-case.

Strcpy copies string from to to, stopping after the null character has been moved. Strncpy copies exactly count characters, appending nulls if from is less than count characters in length; the target may not be null-terminated if the length of from is count or more. Both return to.

Strlen returns the number of non-null characters in s.

Index (rindex) returns a pointer to the first (last) occurrence of character c in string s or zero if c does not occur in the string. Setting c to NULL works.

NAME
     strpbrk - locate multiple characters in string

SYNOPSIS
```
#include <string.h>

char *
strpbrk(s, charset)
char *s;
char *charset;
```

DESCRIPTION
     The strpbrk() function locates in the null-terminated string s the first occurrence of any character in the string charset and returns a pointer to this character. If no characters from charset occur anywhere in s strpbrk() returns NULL.

SEE ALSO
     index(3), memchr(3), rindex(3), strchr(3), strcspn(3), strrchr(3), strsep(3), strspn(3), strstr(3), strtok(3)

STANDARDS
     The strpbrk() function conforms to ANSI C X3.159-1989)``ANSI C'').

NAME
     strsep - separate strings

SYNOPSIS
```
     #include <string.h>

     char *
     strsep(stringp, delim)
     char **stringp;
     char *delim;
```

DESCRIPTION
     The strsep() function locates, in the string referenced by *stringp , the first occurrence of any character in the string delim (or the terminating `\0' character) and replaces it with a `\0'. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in *stringp . The original value of *stringp is returned.

     An ``empty'' field, i.e. one caused by two adjacent delimiter characters, can be detected by comparing the location referenced by the pointer returned in *stringp to `\0'.

     If *stringp is initially NULL, strsep() returns NULL.

EXAMPLES
     The following uses strsep() to parse a string, containing tokens delimited by white space, into an argument vector:

```
     char **ap, *argv[10], *inputstring;

     for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
        if (**ap != '\0')
             ++ap;
```

HISTORY
     The strsep() function is intended as a replacement for the strtok() function. While the strtok() function should be preferred for portability reasons (it conforms to ANSI C X3.159-1989 (``ANSI C'')) it is unable to handle empty fields, i.e. detect fields delimited by two adjacent delimiter characters, or to be used for more than a single string at a time. The strsep() function first appeared in 4.4BSD.

```
NAME
     strspn - span a string

SYNOPSIS
     #include <string.h>

     size_t
     strspn(s, charset)
     char *s;
     char *charset;

DESCRIPTION
     The strcspn() function spans the initial part of the null-
     terminated string s as long as the characters from s occur
     in string charset .

RETURN VALUES
     The strspn() function returns the number of characters
     spanned.

SEE ALSO
     index(3), memchr(3), rindex(3), strchr(3), strcspn(3),
     strpbrk(3), strrchr(3), strsep(3), strstr(3), strtok(3)

STANDARDS
     The strspn() function conforms to ANSI C X3.159-9189 (``ANSI
     C'').
```

NAME

     strstr - locate a substring in a string

SYNOPSIS

```
     #include <string.h>

     char *
     strstr(big, little)
     char *big, *little;
```

DESCRIPTION

The strstr() function locates the first occurrence of the null-terminated string little in the null-terminated string big. If little is the empty string, strstr() returns big; if little occurs nowhere in big, strstr() returns NULL; otherwise strstr() returns a pointer to the first character of the first occurrence of little.

SEE ALSO

index(3), memchr(3), rindex(3), strchr(3), strcspn(3), strpbrk(3), strrchr(3), strsep(3), strspn(3), strtok(3)

STANDARDS

The strstr() function conforms to ANSI C X3/159-1989 (``ANSI C'').

NAME
     strtok - string tokens

SYNOPSIS
```
#include <string.h>

char *
strtok(str, sep)
char *str;
char *sep;
```

DESCRIPTION
     This interface is obsoleted by strsep(3).

     The strtok() function is used to isolate sequential tokens in a null-terminated string, str . These tokens are separated in the string by at least one of the characters in sep . The first time that strtok() is called, str should be specified; subsequent calls, wishing to obtain further tokens from the same string, should pass a null pointer instead. The separator string, sep , must be supplied each time, and may change between calls.

     The strtok() function returns a pointer to the beginning of each subsequent token in the string, after replacing the token itself with a NUL character. When no more tokens remain, a null pointer is returned.

SEE ALSO
     index(3), memchr(3), rindex(3), strchr(3), strcspn(3), strpbrk(3), strrchr(3), strsep(3), strspn(3), strstr(3

STANDARDS
     The strtok() function conforms to ANSI C X3.159-1989 (``ANSI C'').

BUGS
     There is no way to get tokens from multiple strings simul- taneously.

     The System V strtok(), if handed a string containing only delimiter characters, will not alter the next starting point, so that a call to strtok() with a different (or empty) delimiter string may return a non-NULL value. Since this implementation always alters the next starting point, such a sequence of calls would always return NULL.

```
NAME
     strtol - convert string value to a long

SYNOPSIS
     #include <stdlib.h>
     #include <limits.h>

     long
     strtol(nptr, endptr, base)
     char *nptr;
     char **endptr;
     int base;

DESCRIPTION
     The strtol() function converts the string in nptr to a long
     value.  The conversion is done according to the given base,
     which must be between 2 and 36 inclusive, or be the special
     value 0.

     The string may begin with an arbitrary amount of white space
     (as determined by isspace(3)) followed by a single optional
     `+' or `-' sign.  If base is zero or 16, the string may then
     include a `0x' prefix, and the number will be read in base
     16; otherwise, a zero base is taken as 10 (decimal) unless
     the next character is `0', in which case it is taken as 8
     (octal).

     The remainder of the string is converted to a long value in
     the obvious manner, stopping at the first character which is
     not a valid digit in the given base.  (In bases above 10,
     the letter `A' in either upper or lower case represents 10,
     `B' represents 11, and so forth, with `Z' representing 35.)

     If endptr is non nil, strtol() stores the address of the
     first invalid character in *endptr . If there were no digits
     at all, however, strtol() stores the original value of nptr
     in *endptr.  (Thus, if *nptr is not `\0' but **endptr is
     `\0' on return, the entire string was valid.)

RETURN VALUES
     The strtol() function returns the result of the conversion,
     unless the value would underflow or overflow.  If an under-
     flow occurs, strtol() returns LONG_MIN.  If an overflow
     occurs, strtol() returns LONG_MAX.  In both cases, errno is
     set to ERANGE .

ERRORS
     [ERANGE]        The given string was out of range; the value
               converted has been clamped.

SEE ALSO
     atof(3), atoi(3), atol(3), strtoul(3)
```

STANDARDS

The strtol() function conforms to ANSI C X3.159-1989 (``ANSI C'').

BUGS

Ignores the current locale.

NAME
     strtoul - convert a string to an unsigned long

SYNOPSIS
```
     #include <stdlib.h>
     #include <limits.h>

     unsigned long
     strtoul(nptr, endptr, base)
     char *nptr;
     char **endptr;
     int base;
```

DESCRIPTION

The strtoul() function converts the string in nptr to an unsigned long value. The conversion is done according to the given base, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by isspace(3)) followed by a single optional \`+' or \`-' sign. If base is zero or 16, the string may then include a \`0x' prefix, and the number will be read in base 16; otherwise, a zero base is taken as 10 (decimal) unless the next character is \`0', in which case it is taken as 8 (octal).

The remainder of the string is converted to an unsigned long value in the obvious manner, stopping at the end of the string or at the first character that does not produce a valid digit in the given base. (In bases above 10, the letter \`A' in either upper or lower case represents 10, \`B' represents 11, and so forth, with \`Z' representing 35.)

If endptr is non nil, strtoul() stores the address of the first invalid character in *endptr . If there were no digits at all, however, strtoul() stores the original value of nptr in *endptr . (Thus, if *nptr is not \`\0' but **endptr is \`\0' on return, the entire string was valid.)

RETURN VALUES

The strtoul() function returns either the result of the conversion or, if there was a leading minus sign, the negation of the result of the conversion, unless the original (non-negated) value would overflow; in the latter case, strtoul() returns ULONG_MAX and sets the global variable errno to ERANGE .

ERRORS

[ERANGE]   The given string was out of range; the value converted has been clamped.

SEE ALSO
     strtol(3)

STANDARDS
     The strtoul() function conforms to ANSI C X3.159-1989
     (``ANSI C'').

BUGS
     Ignores the current locale.

NAME
     stty, gtty - set and get terminal state (defunct)

SYNOPSIS

```
#include <sgtty.h>

stty(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;
```

DESCRIPTION
     This interface is obsoleted by ioctl(2).

     Stty sets the state of the terminal associated with fd. Gtty retrieves the state of the terminal associated with fd. To set the state of a terminal the call must have write permission.

     The stty call is actually ``ioctl(fd, TIOCSETP, buf)'', while the gtty call is ``ioctl(fd, TIOCGETP, buf)''. See ioctl(2) and tty(4) for an explanation.

DIAGNOSTICS
     If the call is successful 0 is returned, otherwise -1 is returned and the global variable errno contains the reason for the failure.

SEE ALSO
     ioctl(2), tty(4)

NAME
     swab - swap bytes

SYNOPSIS
```
     swab(from, to, nbytes)
     char *from, *to;
```

DESCRIPTION
     Swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. Nbytes should be even.

NAME
     sysctl - get or set system information

SYNOPSIS
```
     #include <sys/sysctl.h>

     int
     sysctl(name, namelen, oldp, *oldlenp, *newp, newlen)
      int *name;
      u_int namelen;
      void *oldp;
      size_t *oldlenp;
      void *newp;
      size_t newlen
```

DESCRIPTION
     The sysctl function retrieves system information and allows processes with appropriate privileges to set system information. The information available from sysctl consists of integers, strings, and tables. Information may be retrieved and set from the command interface using the sysctl(1) utility.

     Unless explicitly noted below, sysctl returns a consistent snapshot of the data requested. Calls to sysctl are serialized to avoid deadlock.

     The state is described using a ``Management Information Base'' (MIB) style name, listed in name , which is a namelen length array of integers.

     The information is copied into the buffer specified by oldp .   The size of the buffer is given by the location specified by oldlenp before the call, and that location gives the amount of data copied after a successful call. If the amount of data available is greater than the size of the buffer supplied, the call supplies as much data as fits in the buffer provided and returns with the error code ENOMEM. If the old value is not desired, oldp and oldlenp should be set to NULL.

     The size of the available data can be determined by calling sysctl with a NULL parameter for oldp. The size of the available data will be returned in the location pointed to by oldlenp. For some operations, the amount of space may change often. For these operations, the system attempts to round up so that the returned size is large enough for a call to return the data shortly thereafter.

     To set a new value, newp is set to point to a buffer of length newlen from which the requested value is to be taken. If a new value is not to be set, newp should be set to NULL and newlen set to 0.

The top level names are defined with a CTL_ prefix in <sys/sysctl.h>, and are as follows. The next and subsequent levels down are found in the include files listed here, and described in separate sections below.

Name	Next level names	Description
CTL_DEBUG	sys/sysctl.h	Debugging
CTL_FS	sys/sysctl.h	File system
CTL_HW	sys/sysctl.h	Generic CPU, I/O
CTL_KERN	sys/sysctl.h	High kernel limits
CTL_MACHDEP	sys/sysctl.h	Machine dependent
CTL_NET	sys/socket.h	Networking
CTL_USER	sys/sysctl.h	User-level
CTL_VM	vm/vm_param.h	Virtual memory

For example, the following retrieves the maximum number of processes allowed in the system:

```
int mib[2], maxproc;
size_t len;

mib[0] = CTL_KERN;
mib[1] = KERN_MAXPROC;
len = sizeof(maxproc);
sysctl(mib, 2, &maxproc, &len, NULL, 0);
```

To retrieve the standard search path for the system utilities:

```
int mib[2];
size_t len;
char *p;

mib[0] = CTL_USER;
mib[1] = USER_CS_PATH;
sysctl(mib, 2, NULL, &len, NULL, 0);
p = malloc(len);
sysctl(mib, 2, p, &len, NULL, 0);
```

CTL_DEBUG

The debugging variables vary from system to system. A debugging variable may be added or deleted without need to recompile sysctl to know about it. Each time it runs, sysctl gets the list of debugging variables from the kernel and displays their current values. The system defines twenty struct ctldebug variables named debug0 through debug19. They are declared as separate variables so that they can be individually initialized at the location of their associated variable. The loader prevents multiple use of the same variable by issuing errors if a variable is initialized in more than one place. For example, to export the variable dospecialcheck as a debugging variable, the following declaration would be used:

```
int dospecialcheck = 1;
struct ctldebug debug5 = { "dospecialcheck", &dospecialcheck };
```

CTL_FS
There are currently no second level names for the file system.

CTL_HW
The string and integer information available for the CTL_HW level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
HW_MACHINE	string	no
HW_MODEL	string	no
HW_NCPU	integer	no
HW_BYTEORDER	integer	no
HW_PHYSMEM	integer	no
HW_USERMEM	integer	no
HW_PAGESIZE	integer	no

HW_MACHINE
The machine class.

HW_MODEL
The machine model

HW_NCPU
The number of cpus.

HW_BYTEORDER
The byteorder (3412, 4321, or 1234).

HW_PHYSMEM
The bytes of physical memory.

HW_USERMEM
The bytes of non-kernel memory.

HW_PAGESIZE
The software page size.

CTL_KERN
The string and integer information available for the CTL_KERN level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value. The types of data currently available are

process information, system inodes, the open file entries, routing table entries, virtual memory statistics, load average history, and clock rate information.

Second level name	Type	Changeable
KERN_ARGMAX	integer	no
KERN_BOOTTIME	struct timeval	no
KERN_CHOWN_RESTRICTED	integer	no
KERN_CLOCKRATE	struct clockinfo	no
KERN_FILE	struct file	no
KERN_HOSTID	long	yes
KERN_HOSTNAME	string	yes
KERN_JOB_CONTROL	integer	no
KERN_MAXFILES	integer	no
KERN_MAXPROC	integer	no
KERN_MAXINODES	integer	no
KERN_MAXTEXTS	integer	no
KERN_NGROUPS	integer	no
KERN_OSRELEASE	string	no
KERN_OSREV	integer	no
KERN_OSTYPE	string	no
KERN_POSIX1	integer	no
KERN_PROC	struct proc	no
KERN_PROF	node	not applicable
KERN_SAVED_IDS	integer	no
KERN_SECURELVL	integer	raise only
KERN_TEXT	struct text	no
KERN_VERSION	string	no
KERN_INODE	struct inode	no

KERN_ARGMAX
The maximum bytes of argument to exec(2).

KERN_BOOTTIME
A struct timeval structure is returned. This structure contains the time that the system was booted.

KERN_CLOCKRATE
A struct clockinfo structure is returned. This structure contains the clock, statistics clock and profiling clock frequencies, and the number of micro-seconds per hz tick.

KERN_FILE
Return the entire file table as an array of extended file structures. Each element of the array contains the kernel address of a file struct inode * followed by the file itself struct file. There can never be more than KERN_MAXFILES inodes returned.

KERN_HOSTID

Get or set the host id.

KERN_HOSTNAME
Get or set the hostname.

KERN_JOB_CONTROL
Return 1 if job control is available on this system, otherwise 0.

KERN_MAXFILES
The maximum number of open files that may be open in the system.

KERN_MAXPROC
The maximum number of simultaneous processes the system will allow.

KERN_MAXINODES
The maximum number of inodes available on the system.

KERN_MAXTEXTS
The maximum number of text structures available on the system.

KERN_NGROUPS
The maximum number of supplemental groups.

KERN_OSRELEASE
The system release string.

KERN_OSREV
The system revision string.

KERN_OSTYPE
The system type string.

KERN_POSIX1
The version of ISO/IEC 9945 (POSIX 1003.1) with which the system attempts to comply.

KERN_PROC
Return the entire process table, or a subset of it. An array of struct kinfo_proc structures is returned, whose size depends on the current number of such objects in the system.

The third and fourth level names are as follows:

Third level name	Fourth level is:
KERN_PROC_ALL	None
KERN_PROC_PID	A process ID
KERN_PROC_PGRP	A process group

KERN_PROC_TTY	A tty device
KERN_PROC_UID	A user ID
KERN_PROC_RUID	A real user ID
KERN_PROF	Return kernel profiling information.

If the kernel is not compiled for profiling, attempts to retrieve any of the KERN_PROF values will fail with EOPNOTSUPP.

The third level names for the string and integer profiling information is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Third level name	Type	Changeable
GPROF_STATE	integer	yes
GPROF_COUNT	u_short[]	yes
GPROF_FROMS	u_short[]	yes
GPROF_TOS	struct tostruct	yes
GPROF_GMONPARAM	struct gmonparam	no

The variables are as follows:

GPROF_STATE
Returns GMON_PROF_ON or GMON_PROF_OFF to show that profiling is running or stopped.

GPROF_COUNT
Array of statistical program counter counts.

GPROF_FROMS
Array indexed by program counter of call-from points.

GPROF_TOS
Array of struct tostruct describing destination of calls and their counts.

GPROF_GMONPARAM
Structure giving the sizes of the above arrays.

KERN_SAVED_IDS
Returns 1 if saved set-group and saved set-user ID is available.

KERN_SECURELVL
The system security level. This level may be raised by processes with appropriate privilege. It may only be lowered by process 1.

KERN_VERSION
The system version string.

KERN_INODE
Return the entire inode table. Note, the inode table is not

necessarily a consistent snapshot of the system. The returned data consists of an array whose size depends on the current number of such objects in the system. Each element of the array contains the kernel address of a inode struct inode * followed by the inode itself struct inode. There can never be more than KERN_MAXINODES inodes returned.

KERN_TEXT
Return the entire text table. The returned data consists of an array whose size depends on the current number of such objects active in the system. Each element of the array contains the kernel address of a text struct text * followed by the text structure itself struct text. There can never be more structures than returned by KERN_MAXTEXTS.

CTL_MACHDEP
The set of variables defined is architecture dependent. Most architectures define at least the following variables.

Second level name	Type	Changeable
CPU_CONSDEV	dev_t	no

CTL_NET
The string and integer information available for the CTL_NET level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
PF_ROUTE	routing messages	no
PF_INET	Internet values	yes

PF_ROUTE
Return the entire routing table or a subset of it. The data is returned as a sequence of routing messages (see route(4) for the header file, format and meaning). The length of each message is contained in the message header.

The third level name is a protocol number, which is currently always 0. The fourth level name is an address family, which may be set to 0 to select all address families. The fifth and sixth level names are as follows:

Fifth level name	Sixth level is:
NET_RT_FLAGS	rtflags
NET_RT_DUMP	None
NET_RT_IFLIST	None

PF_INET
Get or set various global information about the internet protocols. The third level name is the protocol.

The fourth level name is the variable name. The currently defined protocols and names are:

Protocol name	Variable name	Type	Changeable
ip	forwarding	integer	yes
ip	redirect	integer	yes
ip	ttl	integer	yes
icmp	maskrepl	integer	yes
udp	checksum	integer	yes

The variables are as follows:

ip.forwarding
Returns 1 when IP forwarding is enabled for the host, meaning that the host is acting as a router.

ip.redirect
Returns 1 when ICMP redirects may be sent by the host. This option is ignored unless the host is routing IP packets, and should normally be enabled on all systems.

ip.ttl
The maximum time-to-live (hop count) value for an IP packet sourced by the system. This value applies to normal transport protocols, not to ICMP.

icmp.maskrepl
Returns 1 if ICMP network mask requests are to be answered.

udp.checksum
Returns 1 when UDP checksums are being computed and checked. Disabling UDP checksums is strongly discouraged.

CTL_USER
The string and integer information available for the CTL_USER level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
USER_BC_BASE_MAX	integer	no
USER_BC_DIM_MAX	integer	no
USER_BC_SCALE_MAX	integer	no
USER_BC_STRING_MAX	integer	no
USER_COLL_WEIGHTS_MAX	integer	no
USER_CS_PATH	string	no
USER_EXPR_NEST_MAX	integer	no
USER_LINE_MAX	integer	no
USER_POSIX2_CHAR_TERM	integer	no

USER_POSIX2_C_BIND	integer	no
USER_POSIX2_C_DEV	integer	no
USER_POSIX2_FORT_DEV	integer	no
USER_POSIX2_FORT_RUN	integer	no
USER_POSIX2_LOCALEDEF	integer	no
USER_POSIX2_SW_DEV	integer	no
USER_POSIX2_UPE	integer	no
USER_POSIX2_VERSION	integer	no
USER_RE_DUP_MAX	integer	no
USER_STREAM_MAX	integer	no
USER_TZNAME_MAX	integer	no

USER_BC_BASE_MAX
The maximum ibase/obase values in the bc(1) utility.

USER_BC_DIM_MAX
The maximum array size in the bc(1) utility.

USER_BC_SCALE_MAX
The maximum scale value in the bc(1) utility.

USER_BC_STRING_MAX
The maximum string length in the bc(1) utility.

USER_COLL_WEIGHTS_MAX
The maximum number of weights that can be assigned to any entry of the LC_COLLATE order keyword in the locale definition file.

USER_CS_PATH
Return a value for the PATH environment variable that finds all the standard utilities.

USER_EXPR_NEST_MAX
The maximum number of expressions that can be nested within parenthesis by the expr(1) utility.

USER_LINE_MAX
The maximum length in bytes of a text-processing utility's input line.

USER_POSIX2_CHAR_TERM
Return 1 if the system supports at least one terminal type capable of all operations described in POSIX 1003.2, otherwise 0.

USER_POSIX2_C_BIND
Return 1 if the system's C-language development facilities support the C-Language Bindings Option, otherwise 0.

USER_POSIX2_C_DEV

Return 1 if the system supports the C-Language Development Utilities Option, otherwise 0.

USER_POSIX2_FORT_DEV
Return 1 if the system supports the FORTRAN Development Utilities Option, otherwise 0.

USER_POSIX2_FORT_RUN
Return 1 if the system supports the FORTRAN Runtime Utilities Option, otherwise 0.

USER_POSIX2_LOCALEDEF
Return 1 if the system supports the creation of locales, otherwise 0.

USER_POSIX2_SW_DEV
Return 1 if the system supports the Software Development Utilities Option, otherwise 0.

USER_POSIX2_UPE
Return 1 if the system supports the User Portability Utilities Option, otherwise 0.

USER_POSIX2_VERSION
The version of POSIX 1003.2 with which the system attempts to comply.

USER_RE_DUP_MAX
The maximum number of repeated occurrences of a regular expression permitted when using interval notation.

USER_STREAM_MAX
The minimum maximum number of streams that a process may have open at any one time.

USER_TZNAME_MAX
The minimum maximum number of types supported for the name of a timezone.

CTL_VM
The string and integer information available for the CTL_VM level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
VM_LOADAVG	struct loadavg	no
VM_METER	struct vmtotal	no
VM_SWAPMAP	struct map	no
VM_COREMAP	struct map	no

```
     VM_LOADAVG
        Return the load average history.  The returned data
        consists of a struct loadavg.

     VM_METER
        Return the system wide virtual memory statistics.  The
        returned data consists of a struct vmtotal.

     VM_SWAPMAP
        Return the swapmap.  The size of this structure is
        fixed and may be determined by specifying a oldlenp
        initialized to zero, the kernel will fill in the size
        of the swapmap.

     VM_COREMAP
        Same as for swapmap above except that the core alloca-
        tion map is returned.

RETURN VALUES
     If the call to sysctl is successful, 0 is returned.  Other-
     wise -1 is returned and errno is set appropriately.

ERRORS
     The following errors may be reported:

     EFAULT     The buffer name, oldp , newp , or length
                pointer oldlenp contains an invalid address.

     EINVAL     The name array is less than two or greater
                than CTL_MAXNAME.

     EINVAL     A non-null newp is given and its specified
                length in newlen is too large or too small.

     ENOMEM     The length pointed to by oldlenp is too short
                to hold the requested value.

     ENOTDIR         The name array specifies an intermediate
                rather than terminal name.

     EOPNOTSUPP     The name array specifies a value that is unk-
                nown.

     EPERM      An attempt is made to set a read-only value.

     EPERM      A process without appropriate privilege
                attempts to set a value.

FILES
     <sys/sysctl.h> definitions for top level identifiers, second
                level kernel and hardware identifiers, and
                user level identifiers
     <sys/socket.h> definitions for second level network identif-
               iers

     <sys/gmon.h>   definitions for third level profiling iden-
               tifiers

     <sys/vmparam.h>
               definitions for second level virtual memory
               identifiers

     <netinet/in.h> definitions for third level Internet identif-
               iers and fourth level IP identifiers

     <netinet/icmp_var.h>
               definitions for fourth level ICMP identifiers

     <netinet/udp_var.h>
               definitions for fourth level UDP identifiers
```

SEE ALSO

sysctl(8)

HISTORY

The sysctl function first appeared in 4.4BSD.

The KERN_TEXT, KERN_MAXTEXTS, VM_SWAPMAP, VM_COREMAP options are 2.11BSD specific extensions to the 4.4BSD sysctl impl- mentation.

Having KERN_FILE return the address of the file structure before the actual struct file is a 2.11BSD enhancement. The inode (vnode under 4.4) table was handled this way.

NAME
     syserrlst, __errlst - read system error messages from file

SYNOPSIS
```
     char *
     syserrlst(err)
     int err;

     char *
     __errlst(err, path);
     int err;
     char *path;
```

DESCRIPTION
     Syserrlst(3) reads the error message string corresponding to
     err from the file /etc/syserrlst.

     __errlst(3) reads the error message string corresponding to
     err from the file path. The file path must be in the format
     described in syserrlst(5).

     NULL is returned if err is out of bounds (negative or
     greater than the highest message number in /etc/syserrlst or
     path) or if the error message file can not be opened. It is
     the responsibility of the caller (strerror(3)) to check for
     and properly handle the NULL return.

RETURN VALUE
     NULL if an error was encountered in opening the error message file, if the error was out of bounds, or if the file
     did not start with the correct magic number. Otherwise a
     char * is returned pointing to a static buffer containing
     the text of the error message.

ERRORS
     syserrlst(3) and __errlst(3) can return any of the errors
     for the open(2), lseek(2), or read(2) system calls.

SEE ALSO
     perror(3), strerror(3), syserrlst(5)

HISTORY
     syserrlst(3), and __errlst(3) were created for 2.11BSD with
     the aim of saving 2kb of Data space in programs which called
     perror(3), or strerror(3).

BUGS
     The information is stored in a static buffer.

NAME

syslog , vsyslog , openlog , closelog , setlogmask - control system log

SYNOPSIS

```
#include <syslog.h>
#include <varargs.h>

void syslog(priority, message, ...);
int priority;
char *message;

void vsyslog(priority, message, args);
int priority;
char *message;
va_list args;

void openlog(ident, logopt, facility);
char *ident;
int logopt;
int facility;

void closelog();

int setlogmask(maskpri);
int maskpri;
```

DESCRIPTION

The syslog() function writes message to the system message logger. The message is then written to the system console, log files, logged-in users, or forwarded to other machines as appropriate. (See syslogd(8)).

The message is identical to a printf(3) format string, except that %m is replaced by the current error message as denoted by the global variable errno. See strerror(3)). A trailing newline is added if none is present.

The vsyslog() function is an alternate form in which the arguments have already been captured using the variable-length argument facilities of varargs(3).

The message is tagged with priority. Priorities are encoded as a facility and a level. The facility describes the part of the system generating the message. The level is selected from the following ordered (high to low) list:

LOG_EMERG	A panic condition. This is normally broadcast to all users.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.

LOG_CRIT	Critical conditions, e.g., hard device errors.
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions, but should possibly be handled specially.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

The openlog() function provides for more specialized processing of the messages sent by syslog() and vsyslog(). The parameter ident is a string that will be prepended to every message. The logopt argument is a bit field specifying logging options, which is formed by OR'ing one or more of the following values:

LOG_CONS	If syslog cannot pass the message to syslogd it will attempt to write the message to the console (/dev/console).
LOG_NDELAY	Open the connection to syslogd immediately. Normally the open is delayed until the first message is logged. Useful for programs that need to manage the order in which file descriptors are allocated.
LOG_PERROR	Write the message to standard error output as well to the system log.
LOG_PID	Log the process id with each message: useful for identifying instantiations of daemons.

The facility parameter encodes a default facility to be assigned to all messages that do not have an explicit facility encoded:

LOG_AUTH	The authorization system: login(1), su(1), getty(8), etc.
LOG_AUTHPRIV	The same as LOG_AUTH , but logged to a file readable only by selected individuals.
LOG_CRON	The clock daemon.
LOG_DAEMON	System daemons, such as routed(8), that are not provided for explicitly by other

```
               facilities.

     LOG_KERN        Messages generated by the kernel.  These can-
               not be generated by any user processes.

     LOG_LPR         The line printer spooling system: lpr(1),
               lpc(8), lpd(8), etc.

     LOG_MAIL        The mail system.

     LOG_NEWS        The network news system.

     LOG_SYSLOG     Messages generated internally by syslogd(8).

     LOG_USER        Messages generated by random user processes.
               This is the default facility identifier if
               none is specified.

     LOG_UUCP        The uucp system.

     LOG_LOCAL0     Reserved for local use.  Similarly for
               LOG_LOCAL1 through LOG_LOCAL7.
```

The closelog function can be used to close the log file.

The setlogmask function sets the log priority mask to maskpri and returns the previous mask. Calls to syslog with a priority not set in maskpri are rejected. The mask for an individual priority pri is calculated by the macro LOG_MASK(pri). The mask for all priorities up to and including toppri is given by the macro LOG_UPTO(toppri). The default allows all priorities to be logged.

RETURN VALUES

The routines closelog(), openlog(), syslog() and vsyslog() return no value.

The routine setlogmask() always returns the previous log mask level.

EXAMPLES

```
          syslog(LOG_ALERT, "who: internal error 23");

          openlog("ftpd", LOG_PID, LOG_DAEMON);
          setlogmask(LOG_UPTO(LOG_ERR)); syslog(LOG_INFO,
          "Connection from host %d", CallingHost);

          syslog(LOG_INFO|LOG_LOCAL2, "foobar error: %m");
```

SEE ALSO

logger(1), syslogd(8)

BUGS

Under 2.11BSD the logfile /usr/adm/messages is used if a non networking kernel has been booted. That file must be publically writeable in this case.

HISTORY

These functions appeared in 4.2BSD.

NAME

system - issue a shell command

SYNOPSIS

```
system(string)
char *string;
```

DESCRIPTION

System causes the string to be given to sh(1) as input as if the 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.

SEE ALSO

popen(3S), execve(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs - terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base termcap(5). These are low level routines; see curses(3X) for a higher level package.

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. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type name is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than /etc/termcap. This can speed up entry into programs that call tgetent, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file /etc/termcap.

Tgetnum gets the numeric value of capability id, returning -1 if is not given for the terminal. Tgetflag returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. Tgetstr returns the string value of the capability id, places it in the buffer at area, and advances the area pointer. It decodes the abbreviations for this field described in termcap(5), except for cursor addressing and padding information. Tgetstr returns NULL if the capability was not found.

Tgoto returns a cursor addressing string decoded from cm to go to column destcol in line destline. It uses the external variables UP (from the up capability) and BC (if bc is given rather than bs) if necessary to avoid placing \n, ^D or ^@ in the returned string. (Programs which call tgoto should be sure to turn off the XTABS bit(s), since tgoto may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then tgoto returns "OOPS".

Tputs decodes the leading padding information of the string cp; affcnt gives the number of lines affected by the operation, or 1 if this is not applicable, outc is a routine which is called with each character in turn. The external variable ospeed should contain the output speed of the terminal as encoded by stty(3). The external variable PC should contain a pad character to be used (from the pc capability) if a null (^@) is inappropriate.

FILES

/usr/lib/libtermcap.a	-ltermcap library
/etc/termcap	data base

SEE ALSO

ex(1), curses(3X), termcap(5)

AUTHOR

William Joy

NAME

time, ftime - get date and time

SYNOPSIS

```
long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

DESCRIPTION

These interfaces are obsoleted by gettimeofday(2).

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If tloc is nonnull, the return value is also stored in the place to which tloc points.

The ftime entry fills in a structure pointed to by its argument, as defined by <sys/timeb.h>:

```
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *   @(#)timeb.h7.1 (Berkeley) 6/4/86
 */

/*
 * Structure returned by ftime system call
 */
struct timeb
{
  time_t   time;
  unsigned short millitm;
  short    timezone;
  short    dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

SEE ALSO

date(1), gettimeofday(2), settimeofday(2), ctime(3)

NAME

times - get process times

SYNOPSIS

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

times(buffer)
struct tms *buffer;
```

DESCRIPTION

This interface is obsoleted by getrusage(2).

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by times:

```
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *   @(#)times.h   7.1 (Berkeley) 6/4/86
 */

/*
 * Structure returned by times()
 */
struct tms {
   time_t    tms_utime;       /* user time */
   time_t    tms_stime;       /* system time */
   time_t    tms_cutime;      /* user time, children */
   time_t    tms_cstime;      /* system time, children */
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(1), getrusage(2), wait3(2), time(3)

NAME

ttyname, isatty, ttyslot - find name of a terminal

SYNOPSIS

```
char *ttyname(filedes)

isatty(filedes)

ttyslot()
```

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor filedes (this is a system file descriptor and has nothing to do with the standard I/O FILE typedef).

Isatty returns 1 if filedes is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the ttys(5) file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if filedes does not describe a terminal device in directory `/dev'.

Ttyslot returns 0 if `/etc/ttys' is inaccessible or if it cannot determine the control terminal.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME
     ualarm - schedule signal after specified time

SYNOPSIS
```
     unsigned ualarm(value, interval)
     unsigned value;
     unsigned interval;
```

DESCRIPTION
     This is a simplified interface to setitimer(2).

     Ualarm causes signal SIGALRM, see signal(3C), to be sent to the invoking process in a number of microseconds given by the value argument. Unless caught or ignored, the signal terminates the process.

     If the interval argument is non-zero, the SIGALRM signal will be sent to the process every interval microseconds after the timer expires (e.g. after value microseconds have passed).

     Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time (on the vax) is 2147483647 microseconds.

     The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO
     getitimer(2), setitimer(2), sigpause(2), sigvec(2), signal(3C), sleep(3), alarm(3), usleep(3)

NOTES (PDP-11)
     On the PDP-11, setitimer(2) rounds the number of microseconds up to seconds resolution, therefore ualarm doesn't give you any more resolution than alarm(3).

NAME

uname - get system identification

SYNOPSIS

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

int
uname(name)
struct utsname *name
```

DESCRIPTION

The uname function stores nul-terminated strings of information identifying the current system into the structure referenced by name.

The utsname structure is defined in the <sys/utsname.h> header file, and contains the following members:

sysname	Name of the operating system implementation.
nodename	Network name of this machine.
release	Release level of the operating system.
version	Version level of the operating system.
machine	Machine hardware platform.

RETURN VALUES

If uname is successful, 0 is returned, otherwise, -1 is returned and errno is set appropriately.

ERRORS

The uname function may fail and set errno for any of the errors specified for the library functions sysctl(3).

SEE ALSO

uname(1), sysctl(3)

STANDARDS

The uname function conforms to IEEE Std1003.1-88 (``POSIX'').

HISTORY

The uname function first appeared in 4.4BSD.

NAME

ungetc - push character back into input stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
```

DESCRIPTION

Ungetc pushes the character c back on an input stream. That character will be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered.   Attempts to push EOF are rejected.

Fseek(3S) erases all memory of pushed back characters.

SEE ALSO

getc(3S), setbuf(3S), fseek(3S)

DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.

NAME
     utime - set file times

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

     utime(file, timep)
     char *file;
     time_t timep[2];
```

DESCRIPTION
     This interface is obsoleted by utimes(2).

     The utime call uses the `accessed' and `updated' times in that order from the timep vector to set the corresponding recorded times for file.

     The caller must be the owner of the file or the super-user. The `inode-changed' time of the file is set to the current time.

SEE ALSO
     utimes(2), stat(2)

NAME
     valloc - aligned memory allocator

SYNOPSIS
```
     char *valloc(size)
     unsigned size;
```

DESCRIPTION
     Valloc is obsoleted by the current version of malloc, which aligns page-sized and larger allocations.

     Valloc allocates size bytes aligned on a page boundary. It is implemented by calling malloc(3) with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

DIAGNOSTICS
     Valloc returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

BUGS
     Vfree isn't implemented.

NAME

varargs - variable argument list

SYNOPSIS

```
#include <varargs.h>

function(va_alist)
va_dcl
va_list pvar;
va_start(pvar);
f = va_arg(pvar, type);
va_end(pvar);
```

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as printf(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

va_alist is used in a function header to declare a variable argument list.

va_dcl is a declaration for va_alist. Note that there is no semicolon after va_dcl.

va_list is a type which can be used for the variable pvar, which is used to traverse the list. One such variable must always be declared.

va_start(pvar) is called to initialize pvar to the beginning of the list.

va_arg(pvar, type) will return the next argument in the list pointed to by pvar. Type is the type to which the expected argument will be converted when passed as an argument. In standard C, arguments that are char or short should be accessed as int, unsigned char or unsigned short are converted to unsigned int, and float arguments are converted to double. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

va_end(pvar) is used to finish up.

Multiple traversals, each bracketed by va_start ... va_end, are possible.

EXAMPLE

```
#include <varargs.h>
execl(va_alist)
va_dcl
```

```
        {
             va_list ap;
             char *file;
             char *args[100];
             int argno = 0;

             va_start(ap);
             file = va_arg(ap, char *);
             while (args[argno++] = va_arg(ap, char *))
                ;
             va_end(ap);
             return execv(file, args);
        }
```

BUGS

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, execl passes a 0 to signal the end of the list. Printf can tell how many arguments are supposed to be there by the format.

The macros va_start and va_end may be arbitrarily complex; for example, va_start might contain an opening brace, which is closed by a matching brace in va_end. Thus, they should only be used where they could be placed within a single complex statement.