Stardent

PROGRAMMER'S REFERENCE MANUAL, VOL. II

Stardent Computer Inc.

Part Number: 340-0122-01

Change History

340-0021-02	Original
340-0021-03	Software Release 2.0
340-0122-01	January, 1990

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

CONTENTS

2. System Calls

intro(2) ..... introduction to system calls
accept(2) ..... accept a connection on a socket
access(2) ..... determine accessibility of file
bind(2) ..... bind a name to a socket
connect(2) ..... initiate a connection on a socket
dup(2) ..... duplicate a descriptor
dup2(2)(dup) ..... duplicate a descriptor
flock(2) ..... apply or remove an advisory lock on an open file
fsync(2) ..... synchronize a file's in-core state with that on disk
getdtablesize(2) ..... get descriptor table size
getegid(2)(getgid) ..... get group identity
getgid(2) ..... get group identity
getitimer(2) ..... get/set value of interval timer
getrusage(2) ..... get information about resource utilization
getsockopt(2) ..... get and set options on sockets
killpg(2) ..... send signal to a process group
listen(2) ..... listen for connections on a socket
recv(2) ..... receive a message from a socket
recvfrom(2)(recv) ..... receive a message from a socket
select(2) ..... synchronous I/O multiplexing
send(2) ..... send a message from a socket
sendto(2)(send) ..... send a message from a socket
setitimer(2)(getitimer) ..... get/set value of interval timer
setregid(2) ..... set real and effective group ID
setreuid(2) ..... set real and effective user ID's
setsockopt(2)(getsockopt) ..... get and set options on sockets
shutdown(2) ..... shut down part of a full-duplex connection
sigblock(2) ..... block signals
sigpause(2) ..... atomically release blocked signals and wait for interrupt
sigreturn(2) ..... return from signal
sigsetmask(2) ..... set current signal mask
sigstack(2) ..... set and/or get signal stack context
sigvec(2) ..... software signal facilities
socket(2) ..... create an endpoint for communication

NOTE
Entries of the form *dup2(2)(dup)* indicate that the function listed first is described in the entry for the function given in parentheses.

3. Library Subroutines

intro(3) ..... introduction to C library functions
abort(3) ..... generate a fault
abs(3) ..... integer absolute value
alphasort(3)(scandir) ..... scan a directory
asctime(3)(ctime) ..... convert date and time to ASCII
assert(3) ..... program verification
atof(3) ..... convert ASCII to numbers
atoi(3)(atof) ..... convert ASCII to numbers
atol(3)(atof) ..... convert ASCII to numbers
bcmp(3)(bcopy) ..... bit and byte string operations
bcopy(3) ..... bit and byte string operations
bzero(3)(bcopy) ..... bit and byte string operations
calloc(3)(malloc) ..... memory allocator
closedir(3)(opendir) ..... directory operations
closelog(3)(syslog) ..... control system log
crypt(3) ..... DES encryption
ctime(3) ..... convert date and time to ASCII
ctype(3) ..... character classification macros
dbm_clearerr(3)(ndbm) ..... data base subroutines
dbm_close(3)(ndbm) ..... data base subroutines
dbm_delete(3)(ndbm) ..... data base subroutines
dbm_error(3)(ndbm) ..... data base subroutines
dbm_fetch(3)(ndbm) ..... data base subroutines
dbm_firstkey(3)(ndbm) ..... data base subroutines
dbm_nextkey(3)(ndbm) ..... data base subroutines
dbm_open(3)(ndbm) ..... data base subroutines
dbm_store(3)(ndbm) ..... data base subroutines
dn_comp(3)(resolver) ..... resolver routines
dn_expand(3)(resolver) ..... resolver routines
ecvt(3) ..... output conversion
edata(3)(_end) ..... last locations in program
encrypt(3)(crypt) ..... DES encryption
_end(3) ..... last locations in program
endgrent(3)(getgrent) ..... get group file entry
endpwent(3)(getpwent) ..... get password file entry
endusershell(3)(getusershell) ..... get legal user shells
environ(3)(execl) ..... execute a file
_etext(3)(end) ..... last locations in program
exec(3)(execl) ..... execute a file
execl(3) ..... execute a file
execle(3)(execl) ..... execute a file
execlp(3)(execl) ..... execute a file
exect(3)(execl) ..... execute a file
execv(3)(execl) ..... execute a file
execve(3)(execl) ..... execute a file

execvp(3)(execl) ..... execute a file
exit(3) ..... terminate a process after flushing any pending output
fcvt(3)(ecvt) ..... output conversion
ffs(3)(bcopy) ..... bit and byte string operations
free(3)(malloc) ..... memory allocator
frexp(3) ..... split into mantissa and exponent
gcvt(3)(ecvt) ..... output conversion
getenv(3) ..... value for environment name
getgrent(3) ..... get group file entry
getgrgid(3)(getgrent) ..... get group file entry
getgrnam(3)(getgrent) ..... get group file entry
getlogin(3) ..... get login name
getopt(3) ..... get option letter from argv
getpass(3) ..... read a password
getpwent(3) ..... get password file entry
getpwnam(3)(getpwent) ..... get password file entry
getpwuid(3)(getpwent) ..... get password file entry
getusershell(3) ..... get legal user shells
getwd(3) ..... get current working directory pathname
gmtime(3)(ctime) ..... convert date and time to ASCII
index(3)(strcat) ..... string operations
initstate(3)(random) ..... better random number generator
insque(3) ..... insert/remove element from a queue
isalnum(3)(ctype) ..... character classification macros
isascii(3)(ctype) ..... character classification macros
isatty(3)(ttyname) ..... find name of a terminal
iscntrl(3)(ctype) ..... character classification macros
isdigit(3)(ctype) ..... character classification macros
isgraph(3)(ctype) ..... character classification macros
islower(3)(ctype) ..... character classification macros
isprint(3)(ctype) ..... character classification macros
ispunct(3)(ctype) ..... character classification macros
isspace(3)(ctype) ..... character classification macros
isupper(3)(ctype) ..... character classification macros
isxdigit(3)(ctype) ..... character classification macros
ldexp(3)(frexp) ..... split into mantissa and exponent
localtime(3)(ctime) ..... convert date and time to ASCII
longjmp(3)(setjmp) ..... non-local goto
malloc(3) ..... memory allocator
mktemp(3) ..... make a unique file name
modf(3)(frexp) ..... split into mantissa and exponent
moncontrol(3)(monitor) ..... prepare execution profile
monitor(3) ..... prepare execution profile
monstartup(3)(monitor) ..... prepare execution profile
ndbm(3) ..... data base subroutines
nlist(3) ..... get entries from name list

opendir(3) ..... directory operations
openlog(3)(syslog) ..... control system log
pclose(3)(popen) ..... initiate I/O to/from a process
perror(3) ..... system error messages
popen(3) ..... initiate I/O to/from a process
psignal(3) ..... system signal messages
qsort(3) ..... quicker sort
random(3) ..... better random number generator
rcmd(3) ..... routines for returning a stream to a remote command
readdir(3)(opendir) ..... directory operations
realloc(3)(malloc) ..... memory allocator
re_comp(3)(regexp) ..... regular expression handler
re_exec(3)(regexp) ..... regular expression handler
remque(3)(insque) ..... insert/remove element from a queue
res_init(3)(resolver) ..... resolver routines
res_mkquery(3)(resolver) ..... resolver routines
res_send(3)(resolver) ..... resolver routines
rewinddir(3)(opendir) ..... directory operations
rexec(3) ..... return stream to a remote command
rindex(3)(strcat) ..... string operations
rresvport(3)(rcmd) routines for returning a stream to a remote command
ruserok(3)(rcmd) .. routines for returning a stream to a remote command
scandir(3) ..... scan a directory
seekdir(3)(opendir) ..... directory operations
setegid(3)(setuid) ..... set user and group ID
seteuid(3)(setuid) ..... set user and group ID
setgid(3)(setuid) ..... set user and group ID
setgrent(3)(getgrent) ..... get group file entry
setjmp(3) ..... non-local goto
setkey(3)(crypt) ..... DES encryption
setlogmask(3)(syslog) ..... control system log
setpwent(3)(getpwent) ..... get password file entry
setpwfile(3)(getpwent) ..... get password file entry
setrgid(3)(setuid) ..... set user and group ID
setruid(3)(setuid) ..... set user and group ID
setstate(3)(random) ..... better random number generator
setuid(3) ..... set user and group ID
setusershell(3)(getusershell) ..... get legal user shells
siginterrupt(3) ..... allow signals to interrupt system calls
sleep(3) ..... suspend execution for interval
srandom(3)(random) ..... better random number generator
strcat(3) ..... string operations
strcmp(3)(strcat) ..... string operations
strcpy(3)(strcat) ..... string operations
strlen(3)(strcat) ..... string operations
strncat(3)(strcat) ..... string operations

strncmp(3)(strcat) ..... string operations
strncpy(3)(strcat) ..... string operations
swab(3) ..... swap bytes
sys_errlist(3)(perror) ..... system error messages
syslog(3) ..... control system log
sys_nerr(3)(perror) ..... system error messages
sys_siglist(3)(psignal) ..... system signal messages
system(3) ..... issue a shell command
telldir(3)(opendir) ..... directory operations
timezone(3)(ctime) ..... convert date and time to ASCII
toascii(3)(ctype) ..... character classification macros
tolower(3)(ctype) ..... character classification macros
toupper(3)(ctype) ..... character classification macros
ttyname(3) ..... find name of a terminal
ttyslot(3)(ttyname) ..... find name of a terminal

3C. Compatibility Functions

alarm(3C) ..... schedule signal after specified time
ftime(3C)(time) ..... get date and time
getpw(3C) ..... get name from uid
gtty(3C)(stty) ..... set and get terminal state(defunct)
nice(3C) ..... set program priority
pause(3C) ..... stop until signal
rand(3C) ..... random number generator
signal(3C) ..... simplified software signal facilities
srand(3C)(rand) ..... random number generator
stty(3C) ..... set and get terminal state(defunct)
time(3C) ..... get date and time
times(3C) ..... get process times
utime(3C) ..... set file times
valloc(3C) ..... aligned memory allocator
vtimes(3C) ..... get information about resource utilization

3N. Networking Functions

endhostent(3N)(gethostbyname) ..... get network host entry
endnetent(3N)(getnetent) ..... get network entry
endprotoent(3N)(getprotoent) ..... get protocol entry
endservent(3N)(getservent) ..... get service entry
gethostbyaddr(3N)(gethostbyname) ..... get network host entry
gethostbyname(3N) ..... get network host entry
gethostent(3N)(gethostbyname) ..... get network host entry
getnetbyaddr(3N)(getnetent) ..... get network entry

getnetbyname(3N)(getnetent) ..... get network entry
getnetent(3N) ..... get network entry
getprotobyname(3N)(getprotoent) ..... get protocol entry
getprotobynumber(3N)(getprotoent) ..... get protocol entry
getprotoent(3N) ..... get protocol entry
getservbyname(3N)(getservent) ..... get service entry
getservbyport(3N)(getservent) ..... get service entry
getservent(3N) ..... get service entry
htonl(3N) ..... convert values between host and network byte order
htons(3N)(htonl) .... convert values between host and network byte order
inet(3N) ..... Internet address manipulation routines
inet_addr(3N)(inet) ..... Internet address manipulation routines
inet_lnaof(3N)(inet) ..... Internet address manipulation routines
inet_makeaddr(3N)(inet) ..... Internet address manipulation routines
inet_netof(3N)(inet) ..... Internet address manipulation routines
inet_network(3N)(inet) ..... Internet address manipulation routines
inet_ntoa(3N)(inet) ..... Internet address manipulation routines
ntohl(3N)(htonl) ..... convert values between host and network byte order
ntohs(3N)(htonl) .... convert values between host and network byte order
sethostent(3N)(gethostbyname) ..... get network host entry
setnetent(3N)(getnetent) ..... get network entry
setprotoent(3N)(getprotoent) ..... get protocol entry
setservent(3N)(getservent) ..... get service entry

3S. Standard I/O Functions

clearerr(3S)(ferror) ..... stream status inquiries
fclose(3S) ..... close or flush a stream
fdopen(3S)(fopen) ..... open a stream
feof(3S)(ferror) ..... stream status inquiries
ferror(3S) ..... stream status inquiries
fflush(3S)(fclose) ..... close or flush a stream
fgetc(3S)(getc) ..... get character or word from stream
fgets(3S)(gets) ..... get a string from a stream
fileno(3S)(ferror) ..... stream status inquiries
fopen(3S) ..... open a stream
fprintf(3S)(printf) ..... formatted output conversion
fputc(3S)(putc) ..... put character or word on a stream
fputs(3S)(puts) ..... put a string on a stream
fread(3S) ..... buffered binary input/output
freopen(3S)(fopen) ..... open a stream
fscanf(3S)(scanf) ..... formatted input conversion
fseek(3S) ..... reposition a stream
ftell(3S)(fseek) ..... reposition a stream
fwrite(3S)(fread) ..... buffered binary input/output

getc(3S) ..... get character or word from stream
getchar(3S)(getc) ..... get character or word from stream
gets(3S) ..... get a string from a stream
getw(3S)(getc) ..... get character or word from stream
printf(3S) ..... formatted output conversion
putc(3S) ..... put character or word on a stream
putchar(3S)(putc) ..... put character or word on a stream
puts(3S) ..... put a string on a stream
putw(3S)(putc) ..... put character or word on a stream
rewind(3S)(fseek) ..... reposition a stream
scanf(3S) ..... formatted input conversion
setbuf(3S) ..... assign buffering to a stream
setbuffer(3S)(setbuf) ..... assign buffering to a stream
setlinebuf(3S)(setbuf) ..... assign buffering to a stream
sprintf(3S)(printf) ..... formatted output conversion
sscanf(3S)(scanf) ..... formatted input conversion
stdio(3S) ..... standard buffered input/output package
ungetc(3S) ..... push character back into input stream

3X. Miscellaneous Functions

curses(3X) ..... screen functions with "optimal" cursor motion
dbminit(3X) ..... data base subroutines
delete(3X)(dbminit) ..... data base subroutines
fetch(3X)(dbminit) ..... data base subroutines
firstkey(3X)(dbminit) ..... data base subroutines
nextkey(3X)(dbminit) ..... data base subroutines
store(3X)(dbminit) ..... data base subroutines
tgetent(3X) ..... terminal independent operation routines
tgetflag(3X)(tgetent) ..... terminal independent operation routines
tgetnum(3X)(tgetent) ..... terminal independent operation routines
tgetstr(3X)(tgetent) ..... terminal independent operation routines
tgoto(3X)(tgetent) ..... terminal independent operation routines
tputs(3X)(tgetent) ..... terminal independent operation routines

PERMUTED INDEX

abort(3) generate a fault ..... abort(3)
abs(3) integer absolute value ..... abs(3)
abs(3) integer absolute value ..... abs(3)
accept(2) accept a connection on a socket ..... accept(2)
a socket accept(2) accept a connection on ..... accept(2)
of file access(2) determine accessibility ..... access(2)
access(2) determine accessibility of file ..... access(2)
/inet_netof(3n) Internet address manipulation routines ..... inet(3n)
flock(2) apply or remove an advisory lock on an open file ..... flock(2)
specified time alarm(3C) schedule signal after ..... alarm(3C)
valloc(3C) aligned memory allocator ..... valloc(3C)
valloc(3C) aligned memory allocator ..... valloc(3C)
realloc(3) calloc(3) memory allocator malloc(3) free(3) ..... malloc(3)
calls siginterrupt(3) allow signals to interrupt system ..... siginterrupt(3)
scandir(3) alphasort(3) scan a directory ..... scandir(3)
sigstack(2) set and/or get signal stack context ..... sigstack(2)
on an open file flock(2) apply or remove an advisory lock ..... flock(2)
getopt(3) get option letter from argv ..... getopt(3)
convert date and time to ASCII /asctime(3) timezone(3) ..... ctime(3)
atof(3) atoi(3) atol(3) convert ASCII to numbers ..... atof(3)
ctime(3) localtime(3) gmtime(3) asctime(3) timezone(3) convert/ ..... ctime(3)
assert(3) program verification ..... assert(3)
/setbuffer(3S) setlinebuf(3S) assign buffering to a stream ..... setbuf(3S)
ASCII to numbers atof(3) atoi(3) atol(3) convert ..... atof(3)
numbers atof(3) atoi(3) atol(3) convert ASCII to ..... atof(3)
atof(3) atoi(3) atol(3) convert ASCII to numbers ..... atof(3)
signals and wait for/ sigpause(2) atomically release blocked ..... sigpause(2)
dbm_error(3) dbm_clearerr(3) data base subroutines /dbm_nextkey(3) ..... dbm_open(3)
firstkey(3X) nextkey(3X) data base subroutines /delete(3X) ..... dbminit(3X)
byte string operations bcopy(3) bcmp(3) bzero(3) ffs(3) bit and ..... bcopy(3)
bit and byte string operations bcopy(3) bcmp(3) bzero(3) ffs(3) ..... bcopy(3)
/initstate(3) setstate(3) better random number generator;/ ..... random(3)
fread(3S) fwrite(3S) buffered binary input/output ..... fread(3S)
bind(2) bind a name to a socket ..... bind(2)
bind(2) bind a name to a socket ..... bind(2)
bcopy(3) bcmp(3) bzero(3) ffs(3) bit and byte string operations ..... bcopy(3)
sigblock(2) block signals ..... sigblock(2)
sigpause(2) atomically release blocked signals and wait for/ ..... sigpause(2)
fread(3S) fwrite(3S) buffered binary input/output ..... fread(3S)
stdio(3S) standard buffered input/output package ..... stdio(3S)
/setlinebuf(3S) assign buffering to a stream ..... setbuf(3S)
values between host and network byte order /ntohs(3n) convert ..... htonl(3n)
bcmp(3) bzero(3) ffs(3) bit and byte string operations bcopy(3) ..... bcopy(3)

swab(3) swap bytes ..... swab(3)
string/ bcopy(3) bcmp(3) bzero(3) ffs(3) bit and byte ..... bcopy(3)
intro(3) introduction to C library functions ..... intro(3)
malloc(3) free(3) realloc(3) calloc(3) memory allocator ..... malloc(3)
intro(2) introduction to system calls ..... intro(2)
allow signals to interrupt system calls siginterrupt(3) ..... siginterrupt(3)
number generator; routines for changing generators /random ..... random(3)
ungetc(3S) push character back into input stream ..... ungetc(3S)
/toupper(3) tolower(3) toascii(3) character classificatio ..... isalpha(3)
/fgetc(3S) getw(3S) get character or word from stream ..... getc(3S)
/fputc(3S) putw(3S) put character or word on a stream ..... putc(3S)
tolower(3) toascii(3) character classificatio /toupper(3) ..... isalpha(3)
status/ ferror(3S) feof(3S) clearerr(3S) fileno(3S) stream ..... ferror(3S)
fclose(3S) fflush(3S) close or flush a stream ..... fclose(3S)
/seekdir(3) rewinddir(3) closedir(3) directory operations ..... opendir(3)
system log syslog(3) openlog(3) closelog(3) setlogmask(3) control ..... syslog(3)
system(3) issue a shell command ..... system(3)
return stream to a remote command rexec(3) ..... rexec(3)
returning a stream to a remote command /ruserok(3) routines for ..... rcmd(3)
socket(2) create an endpoint for communication ..... socket(2)
on a socket connect(2) initiate a connection ..... connect(2)
accept(2) accept a connection on a socket ..... accept(2)
connect(2) initiate a connection on a socket ..... connect(2)
shut down part of a full-duplex connection shutdown(2) ..... shutdown(2)
listen(2) listen for connections on a socket ..... listen(2)
set and/or get signal stack context sigstack(2) ..... sigstack(2)
closelog(3) setlogmask(3) control system log /openlog(3) ..... syslog(3)
ecvt(3) fcvt(3) gcvt(3) output conversion ..... ecvt(3)
sprintf(3S) formatted output conversion /fprintf(3S) ..... printf(3S)
sscanf(3S) formatted input conversion scanf(3S) fscanf(3S) ..... scanf(3S)
atof(3) atoi(3) atol(3) convert ASCII to numbers ..... atof(3)
/gmtime(3) asctime(3) timezone(3) convert date and time to ASCII ..... ctime(3)
/htons(3n) ntohl(3n) ntohs(3n) convert values between host and/ ..... htonl(3n)
communication socket(2) create an endpoint for ..... socket(2)
encryption crypt(3) setkey(3) encrypt(3) DES ..... crypt(3)
asctime(3) timezone(3) convert/ ctime(3) localtime(3) gmtime(3) ..... ctime(3)
sigsetmask(2) set current signal mask ..... sigsetmask(2)
pathname getwd(3) get current working directory ..... getwd(3)
optimal cursor motion curses(3X) screen functions with ..... curses(3X)
screen functions with optimal cursor motion curses(3X) ..... curses(3X)
/dbm_error(3) dbm_clearerr(3) data base subroutines ..... dbm_open(3)
firstkey(3X) nextkey(3X) data base subroutines /delete(3X) ..... dbminit(3X)
time(3C) ftime(3C) get date and time ..... time(3C)
asctime(3) timezone(3) convert date and time to ASCII /gmtime(3) ..... ctime(3)
/dbm_nextkey(3) dbm_error(3) dbm_clearerr(3) data base/ ..... dbm_open(3)
dbm_store(3)/ dbm_open(3) dbm_close(3) dbm_fetch(3) ..... dbm_open(3)
/dbm_fetch(3) dbm_store(3) dbm_delete(3) dbm_firstkey(3)/ ..... dbm_open(3)
/dbm_firstkey(3) dbm_nextkey(3) dbm_error(3) dbm_clearerr(3) data/ ..... dbm_open(3)
dbm_open(3) dbm_close(3) dbm_fetch(3) dbm_store(3)/ ..... dbm_open(3)
/dbm_store(3) dbm_delete(3) dbm_firstkey(3) dbm_nextkey(3)/ ..... dbm_open(3)
delete(3X) firstkey(3X)/ dbminit(3X) fetch(3X) store(3X) ..... dbminit(3X)
/dbm_delete(3) dbm_firstkey(3) dbm_nextkey(3) dbm_error(3)/ ..... dbm_open(3)
dbm_fetch(3) dbm_store(3)/ dbm_open(3) dbm_close(3) ..... dbm_open(3)
/dbm_close(3) dbm_fetch(3) dbm_store(3) dbm_delete(3)/ ..... dbm_open(3)
set and get terminal state (defunct) stty(3C) gtty(3C) ..... stty(3C)
dbminit(3X) fetch(3X) store(3X) delete(3X) firstkey(3X)/ ..... dbminit(3X)

crypt(3) setkey(3) encrypt(3) DES encryption ..... crypt(3)
dup(2) dup2(2) duplicate a descriptor ..... dup(2)
getdtablesize(2) get descriptor table size ..... getdtablesize(2)
access(2) determine accessibility of file ..... access(2)
scandir(3) alphasort(3) scan a directory ..... scandir(3)
rewinddir(3) closedir(3) directory operations /seekdir(3) ..... opendir(3)
getwd(3) get current working directory pathname ..... getwd(3)
files in-core state with that on disk fsync(2) synchronize a ..... fsync(2)
routines /res_send(3) res_init(3) dn_comp(3) dn_expand(3) resolver ..... res_mkquery(3)
/res_init(3) dn_comp(3) dn_expand(3) resolver routines ..... res_mkquery(3)
descriptor dup(2) dup2(2) duplicate a ..... dup(2)
dup(2) dup2(2) duplicate a descriptor ..... dup(2)
dup(2) dup2(2) duplicate a descriptor ..... dup(2)
conversion ecvt(3) fcvt(3) gcvt(3) output ..... ecvt(3)
program _end(3) _etext(3) _edata(3) last locations in ..... _end(3)
setregid(2) set real and effective group ID ..... setregid(2)
setreuid(2) set real and effective user IDs ..... setreuid(2)
insque(3) remque(3) insert/remove element from a queue ..... insque(3)
crypt(3) setkey(3) encrypt(3) DES encryption ..... crypt(3)
crypt(3) setkey(3) encrypt(3) DES encryption ..... crypt(3)
locations in program _end(3) _etext(3) _edata(3) last ..... _end(3)
/getgrnam(3) setgrent(3) endgrent(3) get group file entry ..... getgrent(3)
/gethostent(3n) sethostent(3n) endhostent(3n) get network host/ . gethostbyname(3n)
/getnetbyname(3n) setnetent(3n) endnetent(3n) get network entry ..... getnetent(3n)
socket(2) create an endpoint for communication ..... socket(2)
entry /setprotoent(3n) endprotoent(3n) get protocol ..... getprotoent(3n)
password/ /getpwnam(3) setpwent(3) endpwent(3) setpwfile(3) get ..... getpwent(3)
/getservbyname(3n) setservent(3n) endservent(3n) get service entry ..... getservent(3n)
getusershell(3) setusershell(3) endusershell(3) get legal user/ ..... getusershell(3)
nlist(3) get entries from name list ..... nlist(3)
endgrent(3) get group file entry /getgrnam(3) setgrent(3) ..... getgrent(3)
endhostent(3n) get network host entry /sethostent(3n) ..... gethostbyname(3n)
endnetent(3n) get network entry /setnetent(3n) ..... getnetent(3n)
endprotoent(3n) get protocol entry /setprotoent(3n) ..... getprotoent(3n)
setpwfile(3) get password file entry /setpwent(3) endpwent(3) ..... getpwent(3)
endservent(3n) get service entry /setservent(3n) ..... getservent(3n)
/exec(3) execve(3) exect(3) environ(3) execute a file ..... execl(3)
getenv(3) value for environment name ..... getenv(3)
sys_errlist(3) sys_nerr(3) system error messages perror(3) ..... perror(3)
locations in program _end(3) _etext(3) _edata(3) last ..... _end(3)
/execle(3) execlp(3) execvp(3) exec(3) execve(3) exect(3)/ ..... execl(3)
execlp(3) execvp(3) exec(3)/ execl(3) execv(3) execle(3) ..... execl(3)
exec(3)/ execl(3) execv(3) execle(3) execlp(3) execvp(3) ..... execl(3)
execl(3) execv(3) execle(3) execlp(3) execvp(3) exec(3)/ ..... execl(3)
file /execvp(3) exec(3) execve(3) exect(3) environ(3) execute a ..... execl(3)
execve(3) exect(3) environ(3) execute a file /execvp(3) exec(3) ..... execl(3)
sleep(3) suspend execution for interval ..... sleep(3)
moncontrol(3) prepare execution profile /monstartup(3) ..... monitor(3)
execvp(3) exec(3)/ execl(3) execv(3) execle(3) execlp(3) ..... execl(3)
/execlp(3) execvp(3) exec(3) execve(3) exect(3) environ(3)/ ..... execl(3)
/execv(3) execle(3) execlp(3) execvp(3) exec(3) execve(3)/ ..... execl(3)
flushing any pending output exit(3) terminate a process after ..... exit(3)
modf(3) split into mantissa and exponent frexp(3) ldexp(3) ..... frexp(3)
re_comp(3) re_exec(3) regular expression handler ..... re_comp(3)
sigvec(2) software signal facilities ..... sigvec(2)
simplified software signal facilities signal(3C) ..... signal(3C)

abort(3) generate a fault ..... abort(3)
flush a stream fclose(3S) fflush(3S) close or ..... fclose(3S)
ecvt(3) fcvt(3) gcvt(3) output conversion ..... ecvt(3)
fopen(3S) freopen(3S) fdopen(3S) open a stream ..... fopen(3S)
stream status/ ferror(3S) feof(3S) clearerr(3S) fileno(3S) ..... ferror(3S)
fileno(3S) stream status/ ferror(3S) feof(3S) clearerr(3S) ..... ferror(3S)
firstkey(3X)/ dbminit(3X) fetch(3X) store(3X) delete(3X) ..... dbminit(3X)
stream fclose(3S) fflush(3S) close or flush a ..... fclose(3S)
bcopy(3) bcmp(3) bzero(3) ffs(3) bit and byte string/ ..... bcopy(3)
or word/ getc(3S) getchar(3S) fgetc(3S) getw(3S) get character ..... getc(3S)
stream gets(3S) fgets(3S) get a string from a ..... gets(3S)
determine accessibility of file access(2) ..... access(2)
setpwfile(3) get password file entry /endpwent(3) ..... getpwent(3)
setgrent(3) endgrent(3) get group file entry /getgrnam(3) ..... getgrent(3)
exect(3) environ(3) execute a file /execvp(3) exec(3) execve(3) ..... execl(3)
an advisory lock on an open file flock(2) apply or remove ..... flock(2)
mktemp(3) make a unique file name ..... mktemp(3)
utime(3C) set file times ..... utime(3C)
ferror(3S) feof(3S) clearerr(3S) fileno(3S) stream status/ ..... ferror(3S)
disk fsync(2) synchronize a files in-core state with that on ..... fsync(2)
ttyname(3) isatty(3) ttyslot(3) find name of a terminal ..... ttyname(3)
/fetch(3X) store(3X) delete(3X) firstkey(3X) nextkey(3X) data/ ..... dbminit(3X)
advisory lock on an open file flock(2) apply or remove an ..... flock(2)
fclose(3S) fflush(3S) close or flush a stream ..... fclose(3S)
exit(3) terminate a process after flushing any pending output ..... exit(3)
open a stream fopen(3S) freopen(3S) fdopen(3S) ..... fopen(3S)
scanf(3S) fscanf(3S) sscanf(3S) formatted input conversion ..... scanf(3S)
/fprintf(3S) sprintf(3S) formatted output conversion ..... printf(3S)
output conversion printf(3S) fprintf(3S) sprintf(3S) formatted ..... printf(3S)
or word on/ putc(3S) putchar(3S) fputc(3S) putw(3S) put character ..... putc(3S)
stream puts(3S) fputs(3S) put a string on a ..... puts(3S)
binary input/output fread(3S) fwrite(3S) buffered ..... fread(3S)
memory allocator malloc(3) free(3) realloc(3) calloc(3) ..... malloc(3)
stream fopen(3S) freopen(3S) fdopen(3S) open a ..... fopen(3S)
into mantissa and exponent frexp(3) ldexp(3) modf(3) split ..... frexp(3)
input conversion scanf(3S) fscanf(3S) sscanf(3S) formatted ..... scanf(3S)
reposition a stream fseek(3S) ftell(3S) rewind(3S) ..... fseek(3S)
in-core state with that on disk fsync(2) synchronize a files ..... fsync(2)
stream fseek(3S) ftell(3S) rewind(3S) reposition a ..... fseek(3S)
time(3C) ftime(3C) get date and time ..... time(3C)
shutdown(2) shut down part of a full-duplex connection ..... shutdown(2)
introduction to C library functions intro(3) ..... intro(3)
motion curses(3X) screen functions with optimal cursor ..... curses(3X)
input/output fread(3S) fwrite(3S) buffered binary ..... fread(3S)
ecvt(3) fcvt(3) gcvt(3) output conversion ..... ecvt(3)
abort(3) generate a fault ..... abort(3)
rand(3C) srand(3C) random number generator ..... rand(3C)
/setstate(3) better random number generator; routines for changing/ ..... random(3)
generator; routines for changing generators /better random number ..... random(3)
getw(3S) get character or word/ getc(3S) getchar(3S) fgetc(3S) ..... getc(3S)
get character or word/ getc(3S) getchar(3S) fgetc(3S) getw(3S) ..... getc(3S)
table size getdtablesize(2) get descriptor ..... getdtablesize(2)
getgid(2) getegid(2) get group identity ..... getgid(2)
name getenv(3) value for environment ..... getenv(3)
identity getgid(2) getegid(2) get group ..... getgid(2)
getgrnam(3) setgrent(3)/ getgrent(3) getgrgid(3) ..... getgrent(3)

setgrent(3)/ getgrent(3) getgrgid(3) getgrnam(3) ..... getgrent(3)
getgrent(3) getgrgid(3) getgrnam(3) setgrent(3)/ ..... getgrent(3)
sethostent(3n)/ gethostbyname(3n) gethostbyaddr(3n) gethostent(3n) .. gethostbyname(3n)
gethostbyaddr(3n) gethostent(3n)/ gethostbyname(3n) ..... gethostbyname(3n)
endhostent(3n)/ /gethostbyaddr(3n) gethostent(3n) sethostent(3n) ..... gethostbyname(3n)
value of interval timer getitimer(2) setitimer(2) get/set ..... getitimer(2)
getlogin(3) get login name ..... getlogin(3)
setnetent(3n)/ getnetent(3n) getnetbyaddr(3n) getnetbyname(3n) ..... getnetent(3n)
getnetent(3n) getnetbyaddr(3n) getnetbyname(3n) setnetent(3n)/ ..... getnetent(3n)
getnetbyname(3n) setnetent(3n)/ getnetent(3n) getnetbyaddr(3n) ..... getnetent(3n)
argv getopt(3) get option letter from ..... getopt(3)
getpass(3) read a password ..... getpass(3)
/getprotobynumber(3n) getprotobyname(3n)/ ..... getprotoent(3n)
getprotoent(3n) getprotobynumber(3n)/ ..... getprotoent(3n)
getprotobynumber(3n)/ getprotoent(3n) ..... getprotoent(3n)
getpw(3C) get name from uid ..... getpw(3C)
getpwnam(3) setpwent(3)/ getpwent(3) getpwuid(3) ..... getpwent(3)
getpwent(3) getpwuid(3) getpwnam(3) setpwent(3)/ ..... getpwent(3)
setpwent(3)/ getpwent(3) getpwuid(3) getpwnam(3) ..... getpwent(3)
about resource utilization getrusage(2) get information ..... getrusage(2)
from a stream gets(3S) fgets(3S) get a string ..... gets(3S)
getservent(3n) getservbyport(3n) getservbyname(3n) setservent(3n)/ ..... getservent(3n)
getservbyname(3n)/ getservent(3n) getservbyport(3n) ..... getservent(3n)
getservbyname(3n) setservent(3n)/ getservent(3n) getservbyport(3n) ..... getservent(3n)
getitimer(2) setitimer(2) get/set value of interval timer ..... getitimer(2)
and set options on sockets getsockopt(2) setsockopt(2) get ..... getsockopt(2)
endusershell(3) get legal user/ getusershell(3) setusershell(3) ..... getusershell(3)
getc(3S) getchar(3S) fgetc(3S) getw(3S) get character or word/ ..... getc(3S)
directory pathname getwd(3) get current working ..... getwd(3)
convert/ ctime(3) localtime(3) gmtime(3) asctime(3) timezone(3) ..... ctime(3)
setjmp(3) longjmp(3) non-local goto ..... setjmp(3)
setgrent(3) endgrent(3) get group file entry /getgrnam(3) ..... getgrent(3)
setrgid(3) set user and group ID /setgid(3) setegid(3) ..... setuid(3)
set real and effective group ID setregid(2) ..... setregid(2)
getgid(2) getegid(2) get group identity ..... getgid(2)
send signal to a process group killpg(2) ..... killpg(2)
state (defunct) stty(3C) gtty(3C) set and get terminal ..... stty(3C)
re_exec(3) regular expression handler re_comp(3) ..... re_comp(3)
/ntohs(3n) convert values between host and network byte order ..... htonl(3n)
endhostent(3n) get network host entry /sethostent(3n) ..... gethostbyname(3n)
ntohs(3n) convert values between/ htonl(3n) htons(3n) ntohl(3n) ..... htonl(3n)
convert values between/ htonl(3n) htons(3n) ntohl(3n) ntohs(3n) ..... htonl(3n)
setrgid(3) set user and group ID /setgid(3) setegid(3) ..... setuid(3)
set real and effective group ID setregid(2) ..... setregid(2)
getgid(2) getegid(2) get group identity ..... getgid(2)
set real and effective user IDs setreuid(2) ..... setreuid(2)
fsync(2) synchronize a files in-core state with that on disk ..... fsync(2)
/tgoto(3X) tputs(3X) terminal independent operation routines ..... tgetent(3X)
/strcpy(3) strncpy(3) strlen(3) index(3) rindex(3) string/ ..... strcat(3)
inet_network(3n) inet_ntoa(3n)/ inet(3n) inet_addr(3n) ..... inet(3n)
inet_ntoa(3n)/ inet(3n) inet_addr(3n) inet_network(3n) ..... inet(3n)
/inet_ntoa(3n) inet_makeaddr(3n) inet_lnaof(3n) inet_netof(3n)/ ..... inet(3n)
/inet_network(3n) inet_ntoa(3n) inet_makeaddr(3n) inet_lnaof(3n)/ ..... inet(3n)
/inet_makeaddr(3n) inet_lnaof(3n) inet_netof(3n) Internet address/ ..... inet(3n)
inet(3n) inet_addr(3n) inet_network(3n) inet_ntoa(3n)/ ..... inet(3n)
/inet_addr(3n) inet_network(3n) inet_ntoa(3n) inet_makeaddr(3n)/ ..... inet(3n)

utilization getrusage(2) get — information about resource ..... getrusage(2)
utilization vtimes(3C) get — information about resource ..... vtimes(3C)
connect(2) — initiate a connection on a socket ..... connect(2)
popen(3) pclose(3) — initiate I/O to/from a process ..... popen(3)
random/ random(3) srandom(3) — initstate(3) setstate(3) better ..... random(3)
fscanf(3S) sscanf(3S) formatted — input conversion scanf(3S) ..... scanf(3S)
push character back into — input stream ungetc(3S) ..... ungetc(3S)
fwrite(3S) buffered binary — input/output fread(3S) ..... fread(3S)
stdio(3S) standard buffered — input/output package ..... stdio(3S)
fileno(3S) stream status — inquiries /feof(3S) clearerr(3S) ..... ferror(3S)
queue insque(3) remque(3) — insert/remove element from a ..... insque(3)
element from a queue — insque(3) remque(3) insert/remove ..... insque(3)
abs(3) — integer absolute value ..... abs(3)
/inet_lnaof(3n) inet_netof(3n) — Internet address manipulation/ ..... inet(3n)
blocked signals and wait for — interrupt /atomically release ..... sigpause(2)
siginterrupt(3) allow signals to — interrupt system calls ..... siginterrupt(3)
sleep(3) suspend execution for — interval ..... sleep(3)
setitimer(2) get/set value of — interval timer getitimer(2) ..... getitimer(2)
calls — intro(2) introduction to system ..... intro(2)
library functions — intro(3) introduction to C ..... intro(3)
functions intro(3) — introduction to C library ..... intro(3)
intro(2) — introduction to system calls ..... intro(2)
select(2) synchronous — I/O multiplexing ..... select(2)
popen(3) pclose(3) initiate — I/O to/from a process ..... popen(3)
/islower(3) isdigit(3) isxdigit(3) — isalnum(3) isspace(3) ispunct(3)/ ..... isalpha(3)
isdigit(3) isxdigit(3)/ — isalpha(3) isupper(3) islower(3) ..... isalpha(3)
/isprint(3) isgraph(3) iscntrl(3) — isascii(3) toupper(3) tolower(3)/ ..... isalpha(3)
a terminal ttyname(3) — isatty(3) ttyslot(3) find name of ..... ttyname(3)
/ispunct(3) isprint(3) isgraph(3) — iscntrl(3) isascii(3) toupper(3)/ ..... isalpha(3)
isalpha(3) isupper(3) islower(3) — isdigit(3) isxdigit(3) isalnum(3)/ ..... isalpha(3)
/isspace(3) ispunct(3) isprint(3) — isgraph(3) iscntrl(3) isascii(3)/ ..... isalpha(3)
isalnum(3)/ isalpha(3) isupper(3) — islower(3) isdigit(3) isxdigit(3) ..... isalpha(3)
/isalnum(3) isspace(3) ispunct(3) — isprint(3) isgraph(3) iscntrl(3)/ ..... isalpha(3)
/isxdigit(3) isalnum(3) isspace(3) — ispunct(3) isprint(3) isgraph(3)/ ..... isalpha(3)
/isdigit(3) isxdigit(3) isalnum(3) — isspace(3) ispunct(3) isprint(3)/ ..... isalpha(3)
system(3) — issue a shell command ..... system(3)
isxdigit(3)/ isalpha(3) — isupper(3) islower(3) isdigit(3) ..... isalpha(3)
/isupper(3) islower(3) isdigit(3) — isxdigit(3) isalnum(3) isspace(3)/ ..... isalpha(3)
process group — killpg(2) send signal to a ..... killpg(2)
mantissa and exponent frexp(3) — ldexp(3) modf(3) split into ..... frexp(3)
/endusershell(3) get — legal user shells ..... getusershell(3)
getopt(3) get option — letter from argv ..... getopt(3)
intro(3) introduction to C — library functions ..... intro(3)
nlist(3) get entries from name — list ..... nlist(3)
socket listen(2) — listen for connections on a ..... listen(2)
on a socket — listen(2) listen for connections ..... listen(2)
timezone(3) convert/ ctime(3) — localtime(3) gmtime(3) asctime(3) ..... ctime(3)
_end(3) _etext(3) _edata(3) last — locations in program ..... _end(3)
apply or remove an advisory — lock on an open file flock(2) ..... flock(2)
setlogmask(3) control system — log /openlog(3) closelog(3) ..... syslog(3)
getlogin(3) get — login name ..... getlogin(3)
setjmp(3) — longjmp(3) non-local goto ..... setjmp(3)
calloc(3) memory allocator — malloc(3) free(3) realloc(3) ..... malloc(3)
/inet_netof(3n) Internet address — manipulation routines ..... inet(3n)
ldexp(3) modf(3) split into — mantissa and exponent frexp(3) ..... frexp(3)
sigsetmask(2) set current signal — mask ..... sigsetmask(2)

valloc(3C) aligned memory allocator ..... valloc(3C)
free(3) realloc(3) calloc(3) memory allocator malloc(3) ..... malloc(3)
recv(2) recvfrom(2) receive a message from a socket ..... recv(2)
send(2) sendto(2) send a message from a socket ..... send(2)
sys_siglist(3) system signal messages psignal(3) ..... psignal(3)
sys_nerr(3) system error messages /sys_errlist(3) ..... perror(3)
mktemp(3) make a unique file name ..... mktemp(3)
exponent frexp(3) ldexp(3) modf(3) split into mantissa and ..... frexp(3)
profile monitor(3) monstartup(3) moncontrol(3) prepare execution ..... monitor(3)
moncontrol(3) prepare execution/ monitor(3) monstartup(3) ..... monitor(3)
prepare execution/ monitor(3) monstartup(3) moncontrol(3) ..... monitor(3)
functions with optimal cursor motion curses(3X) screen ..... curses(3X)
select(2) synchronous I/O multiplexing ..... select(2)
getenv(3) value for environment name ..... getenv(3)
getlogin(3) get login name ..... getlogin(3)
mktemp(3) make a unique file name ..... mktemp(3)
getpw(3C) get name from uid ..... getpw(3C)
nlist(3) get entries from name list ..... nlist(3)
isatty(3) ttyslot(3) find name of a terminal ttyname(3) ..... ttyname(3)
bind(2) bind a name to a socket ..... bind(2)
convert values between host and network byte order /ntohs(3n) ..... htonl(3n)
setnetent(3n) endnetent(3n) get network entry /getnetbyname(3n) ..... getnetent(3n)
/sethostent(3n) endhostent(3n) get network host entry ..... gethostbyname(3n)
/store(3X) delete(3X) firstkey(3X) nextkey(3X) data base subroutines ..... dbminit(3X)
nice(3C) set program priority ..... nice(3C)
list nlist(3) get entries from name ..... nlist(3)
setjmp(3) longjmp(3) non-local goto ..... setjmp(3)
values/ htonl(3n) htons(3n) ntohl(3n) ntohs(3n) convert ..... htonl(3n)
htonl(3n) htons(3n) ntohl(3n) ntohs(3n) convert values between/ ..... htonl(3n)
rand(3C) srand(3C) random number generator ..... rand(3C)
/setstate(3) better random number generator; routines for/ ..... random(3)
atoi(3) atol(3) convert ASCII to numbers atof(3) ..... atof(3)
fopen(3S) freopen(3S) fdopen(3S) open a stream ..... fopen(3S)
or remove an advisory lock on an open file flock(2) apply ..... flock(2)
seekdir(3) rewinddir(3)/ opendir(3) readdir(3) telldir(3) ..... opendir(3)
setlogmask(3) control/ syslog(3) openlog(3) closelog(3) ..... syslog(3)
tputs(3X) terminal independent operation routines /tgoto(3X) ..... tgetent(3X)
ffs(3) bit and byte string operations /bcmp(3) bzero(3) ..... bcopy(3)
closedir(3) directory operations /rewinddir(3) ..... opendir(3)
index(3) rindex(3) string operations /strncpy(3) strlen(3) ..... strcat(3)
curses(3X) screen functions with optimal cursor motion ..... curses(3X)
getopt(3) get option letter from argv ..... getopt(3)
setsockopt(2) get and set options on sockets getsockopt(2) ..... getsockopt(2)
between host and network byte order /ntohs(3n) convert values ..... htonl(3n)
ecvt(3) fcvt(3) gcvt(3) output conversion ..... ecvt(3)
fprintf(3S) sprintf(3S) formatted output conversion printf(3S) ..... printf(3S)
after flushing any pending output /terminate a process ..... exit(3)
standard buffered input/output package stdio(3S) ..... stdio(3S)
shutdown(2) shut down part of a full-duplex connection ..... shutdown(2)
getpass(3) read a password ..... getpass(3)
endpwent(3) setpwfile(3) get password file entry /setpwent(3) ..... getpwent(3)
get current working directory pathname getwd(3) ..... getwd(3)
pause(3C) stop until signal ..... pause(3C)
process popen(3) pclose(3) initiate I/O to/from a ..... popen(3)
a process after flushing any pending output exit(3) terminate ..... exit(3)
sys_nerr(3) system error/ perror(3) sys_errlist(3) ..... perror(3)

to/from a process popen(3) pclose(3) initiate I/O ..... popen(3)
/monstartup(3) moncontrol(3) prepare execution profile ..... monitor(3)
sprintf(3S) formatted output/ printf(3S) fprintf(3S) ..... printf(3S)
nice(3C) set program priority ..... nice(3C)
pending/ exit(3) terminate a process after flushing any ..... exit(3)
killpg(2) send signal to a process group ..... killpg(2)
pclose(3) initiate I/O to/from a process popen(3) ..... popen(3)
times(3C) get process times ..... times(3C)
moncontrol(3) prepare execution profile monitor(3) monstartup(3) ..... monitor(3)
endprotoent(3n) get protocol entry /setprotoent(3n) ..... getprotoent(3n)
signal messages psignal(3) sys_siglist(3) system ..... psignal(3)
stream ungetc(3S) push character back into input ..... ungetc(3S)
puts(3S) fputs(3S) put a string on a stream ..... puts(3S)
/putchar(3S) fputc(3S) putw(3S) put character or word on a stream ..... putc(3S)
putw(3S) put character or word/ putc(3S) putchar(3S) fputc(3S) ..... putc(3S)
put character or word/ putc(3S) putchar(3S) fputc(3S) putw(3S) ..... putc(3S)
on a stream puts(3S) fputs(3S) put a string ..... puts(3S)
a/ putc(3S) putchar(3S) fputc(3S) putw(3S) put character or word on ..... putc(3S)
qsort(3) quicker sort ..... qsort(3)
insert/remove element from a queue insque(3) remque(3) ..... insque(3)
qsort(3) quicker sort ..... qsort(3)
generator rand(3C) srand(3C) random number ..... rand(3C)
rand(3C) srand(3C) random number generator ..... rand(3C)
/initstate(3) setstate(3) better random number generator; routines/ ..... random(3)
setstate(3) better random number/ random(3) srandom(3) initstate(3) ..... random(3)
routines for returning a stream/ rcmd(3) rresvport(3) ruserok(3) ..... rcmd(3)
getpass(3) read a password ..... getpass(3)
rewinddir(3)/ opendir(3) readdir(3) telldir(3) seekdir(3) ..... opendir(3)
setregid(2) set real and effective group ID ..... setregid(2)
setreuid(2) set real and effective user IDs ..... setreuid(2)
allocator malloc(3) free(3) realloc(3) calloc(3) memory ..... malloc(3)
recv(2) recvfrom(2) receive a message from a socket ..... recv(2)
expression handler re_comp(3) re_exec(3) regular ..... re_comp(3)
message from a socket recv(2) recvfrom(2) receive a ..... recv(2)
from a socket recv(2) recvfrom(2) receive a message ..... recv(2)
handler re_comp(3) re_exec(3) regular expression ..... re_comp(3)
re_comp(3) re_exec(3) regular expression handler ..... re_comp(3)
for/ sigpause(2) atomically release blocked signals and wait ..... sigpause(2)
rexec(3) return stream to a remote command ..... rexec(3)
for returning a stream to a remote command /routines ..... rcmd(3)
open file flock(2) apply or remove an advisory lock on an ..... flock(2)
from a queue insque(3) remque(3) insert/remove element ..... insque(3)
fseek(3S) ftell(3S) rewind(3S) reposition a stream ..... fseek(3S)
res_mkquery(3) res_send(3) res_init(3) dn_comp(3)/ ..... res_mkquery(3)
res_init(3) dn_comp(3)/ res_mkquery(3) res_send(3) ..... res_mkquery(3)
dn_comp(3) dn_expand(3) resolver routines /res_init(3) ..... res_mkquery(3)
/get information about resource utilization ..... getrusage(2)
vtimes(3C) get information about resource utilization ..... vtimes(3C)
dn_comp(3)/ res_mkquery(3) res_send(3) res_init(3) ..... res_mkquery(3)
sigreturn(2) return from signal ..... sigreturn(2)
rexec(3) return stream to a remote command ..... rexec(3)
command /ruserok(3) routines for returning a stream to a remote ..... rcmd(3)
fseek(3S) ftell(3S) rewind(3S) reposition a stream ..... fseek(3S)
/readdir(3) telldir(3) seekdir(3) rewinddir(3) closedir(3)/ ..... opendir(3)
remote command rexec(3) return stream to a ..... rexec(3)
/strncpy(3) strlen(3) index(3) rindex(3) string operations ..... strcat(3)

/better random number generator; routines for changing generators ..... random(3)
rcmd(3) rresvport(3) ruserok(3) routines for returning a stream/ ..... rcmd(3)
Internet address manipulation routines /inet_netof(3n) ..... inet(3n)
dn_comp(3) dn_expand(3) resolver routines /res_send(3) res_init(3) ..... res_mkquery(3)
terminal independent operation routines /tgoto(3X) tputs(3X) ..... tgetent(3X)
for returning a stream/ rcmd(3) rresvport(3) ruserok(3) routines ..... rcmd(3)
a stream to/ rcmd(3) rresvport(3) ruserok(3) routines for returning ..... rcmd(3)
scandir(3) alphasort(3) scan a directory ..... scandir(3)
directory scandir(3) alphasort(3) scan a ..... scandir(3)
formatted input conversion scanf(3S) fscanf(3S) sscanf(3S) ..... scanf(3S)
time alarm(3C) schedule signal after specified ..... alarm(3C)
cursor motion curses(3X) screen functions with optimal ..... curses(3X)
opendir(3) readdir(3) telldir(3) seekdir(3) rewinddir(3)/ ..... opendir(3)
multiplexing select(2) synchronous I/O ..... select(2)
send(2) sendto(2) send a message from a socket ..... send(2)
killpg(2) send signal to a process group ..... killpg(2)
from a socket send(2) sendto(2) send a message ..... send(2)
socket send(2) sendto(2) send a message from a ..... send(2)
setservent(3n) endservent(3n) get service entry /getservbyname(3n) ..... getservent(3n)
(defunct) stty(3C) gtty(3C) set and get terminal state ..... stty(3C)
context sigstack(2) set and/or get signal stack ..... sigstack(2)
sigsetmask(2) set current signal mask ..... sigsetmask(2)
utime(3C) set file times ..... utime(3C)
/setsockopt(2) get and set options on sockets ..... getsockopt(2)
nice(3C) set program priority ..... nice(3C)
setregid(2) set real and effective group ID ..... setregid(2)
setreuid(2) set real and effective user IDs ..... setreuid(2)
setgid(3) setegid(3) setrgid(3) set user and group ID /setruid(3) ..... setuid(3)
setlinebuf(3S) assign buffering/ setbuf(3S) setbuffer(3S) ..... setbuf(3S)
assign buffering to a/ setbuf(3S) setbuffer(3S) setlinebuf(3S) ..... setbuf(3S)
/seteuid(3) setruid(3) setgid(3) setegid(3) setrgid(3) set user/ ..... setuid(3)
setegid(3) setrgid(3)/ setuid(3) seteuid(3) setruid(3) setgid(3) ..... setuid(3)
setuid(3) seteuid(3) setruid(3) setgid(3) setegid(3) setrgid(3)/ ..... setuid(3)
file/ /getgrgid(3) getgrnam(3) setgrent(3) endgrent(3) get group ..... getgrent(3)
/gethostbyaddr(3n) gethostent(3n) sethostent(3n) endhostent(3n) get/ gethostbyname(3n)
interval timer getitimer(2) setitimer(2) get/set value of ..... getitimer(2)
goto setjmp(3) longjmp(3) non-local ..... setjmp(3)
encryption crypt(3) setkey(3) encrypt(3) DES ..... crypt(3)
to a/ setbuf(3S) setbuffer(3S) setlinebuf(3S) assign buffering ..... setbuf(3S)
syslog(3) openlog(3) closelog(3) setlogmask(3) control system log ..... syslog(3)
/getnetbyaddr(3n) getnetbyname(3n) setnetent(3n) endnetent(3n) get/ ..... getnetent(3n)
get protocol/ /getprotobyname(3n) setprotoent(3n) endprotoent(3n) ..... getprotoent(3n)
/getpwuid(3) getpwnam(3) setpwent(3) endpwent(3)/ ..... getpwent(3)
entry /setpwent(3) endpwent(3) setpwfile(3) get password file ..... getpwent(3)
effective group ID setregid(2) set real and ..... setregid(2)
effective user IDs setreuid(2) set real and ..... setreuid(2)
/setruid(3) setgid(3) setegid(3) setrgid(3) set user and group ID ..... setuid(3)
setrgid(3)/ setuid(3) seteuid(3) setruid(3) setgid(3) setegid(3) ..... setuid(3)
service entry /getservbyname(3n) setservent(3n) endservent(3n) get ..... getservent(3n)
on sockets getsockopt(2) setsockopt(2) get and set options ..... getsockopt(2)
random(3) srandom(3) initstate(3) setstate(3) better random number/ ..... random(3)
setgid(3) setegid(3) setrgid(3)/ setuid(3) seteuid(3) setruid(3) ..... setuid(3)
get legal user/ getusershell(3) setusershell(3) endusershell(3) ..... getusershell(3)
system(3) issue a shell command ..... system(3)
endusershell(3) get legal user shells /setusershell(3) ..... getusershell(3)
connection shutdown(2) shut down part of a full-duplex ..... shutdown(2)

full-duplex connection shutdown(2) shut down part of a ..... shutdown(2)
sigblock(2) block signals ..... sigblock(2)
interrupt system calls siginterrupt(3) allow signals to ..... siginterrupt(3)
pause(3C) stop until signal ..... pause(3C)
sigreturn(2) return from signal ..... sigreturn(2)
alarm(3C) schedule signal after specified time ..... alarm(3C)
signal(3C) simplified software signal facilities ..... signal(3C)
sigvec(2) software signal facilities ..... sigvec(2)
sigsetmask(2) set current signal mask ..... sigsetmask(2)
psignal(3) sys_siglist(3) system signal messages ..... psignal(3)
sigstack(2) set and/or get signal stack context ..... sigstack(2)
killpg(2) send signal to a process group ..... killpg(2)
signal facilities signal(3C) simplified software ..... signal(3C)
sigblock(2) block signals ..... sigblock(2)
/atomically release blocked signals and wait for interrupt ..... sigpause(2)
siginterrupt(3) allow signals to interrupt system calls ..... siginterrupt(3)
blocked signals and wait for/ sigpause(2) atomically release ..... sigpause(2)
sigreturn(2) return from signal ..... sigreturn(2)
mask sigsetmask(2) set current signal ..... sigsetmask(2)
stack context sigstack(2) set and/or get signal ..... sigstack(2)
facilities sigvec(2) software signal ..... sigvec(2)
facilities signal(3C) simplified software signal ..... signal(3C)
get descriptor table size getdtablesize(2) ..... getdtablesize(2)
interval sleep(3) suspend execution for ..... sleep(3)
bind(2) bind a name to a socket ..... bind(2)
accept a connection on a socket accept(2) ..... accept(2)
initiate a connection on a socket connect(2) ..... connect(2)
listen for connections on a socket listen(2) ..... listen(2)
receive a message from a socket recv(2) recvfrom(2) ..... recv(2)
sendto(2) send a message from a socket send(2) ..... send(2)
communication socket(2) create an endpoint for ..... socket(2)
get and set options on sockets /setsockopt(2) ..... getsockopt(2)
signal(3C) simplified software signal facilities ..... signal(3C)
sigvec(2) software signal facilities ..... sigvec(2)
qsort(3) quicker sort ..... qsort(3)
alarm(3C) schedule signal after specified time ..... alarm(3C)
frexp(3) ldexp(3) modf(3) split into mantissa and exponent ..... frexp(3)
printf(3S) fprintf(3S) sprintf(3S) formatted output/ ..... printf(3S)
rand(3C) srand(3C) random number generator ..... rand(3C)
setstate(3) better/ random(3) srandom(3) initstate(3) ..... random(3)
conversion scanf(3S) fscanf(3S) sscanf(3S) formatted input ..... scanf(3S)
sigstack(2) set and/or get signal stack context ..... sigstack(2)
package stdio(3S) standard buffered input/output ..... stdio(3S)
clearerr(3S) fileno(3S) stream status inquiries /feof(3S) ..... ferror(3S)
input/output package stdio(3S) standard buffered ..... stdio(3S)
pause(3C) stop until signal ..... pause(3C)
dbminit(3X) fetch(3X) store(3X) delete(3X) firstkey(3X)/ ..... dbminit(3X)
strncmp(3) strcpy(3) strncpy(3)/ strcat(3) strncat(3) strcmp(3) ..... strcat(3)
strncpy(3)/ strcat(3) strncat(3) strcmp(3) strncmp(3) strcpy(3) ..... strcat(3)
/strncat(3) strcmp(3) strncmp(3) strcpy(3) strncpy(3) strlen(3)/ ..... strcat(3)
fflush(3S) close or flush a stream fclose(3S) ..... fclose(3S)
get character or word from stream /fgetc(3S) getw(3S) ..... getc(3S)
freopen(3S) fdopen(3S) open a stream fopen(3S) ..... fopen(3S)
put character or word on a stream /fputc(3S) putw(3S) ..... putc(3S)
ftell(3S) rewind(3S) reposition a stream fseek(3S) ..... fseek(3S)
fgets(3S) get a string from a stream gets(3S) ..... gets(3S)

fputs(3S) put a string on a stream puts(3S) ..... puts(3S)
assign buffering to a stream /setlinebuf(3S) ..... setbuf(3S)
/feof(3S) clearerr(3S) fileno(3S) stream status inquiries ..... ferror(3S)
/routines for returning a stream to a remote command ..... rcmd(3)
rexec(3) return stream to a remote command ..... rexec(3)
push character back into input stream ungetc(3S) ..... ungetc(3S)
gets(3S) fgets(3S) get a string from a stream ..... gets(3S)
puts(3S) fputs(3S) put a string on a stream ..... puts(3S)
bzero(3) ffs(3) bit and byte string operations /bcmp(3) ..... bcopy(3)
strlen(3) index(3) rindex(3) string operations /strncpy(3) ..... strcat(3)
/strncmp(3) strcpy(3) strncpy(3) strlen(3) index(3) rindex(3)/ ..... strcat(3)
strcpy(3) strncpy(3)/ strcat(3) strncat(3) strcmp(3) strncmp(3) ..... strcat(3)
strcat(3) strncat(3) strcmp(3) strncmp(3) strcpy(3) strncpy(3)/ ..... strcat(3)
/strcmp(3) strncmp(3) strcpy(3) strncpy(3) strlen(3) index(3)/ ..... strcat(3)
terminal state (defunct) stty(3C) gtty(3C) set and get ..... stty(3C)
dbm_clearerr(3) data base subroutines /dbm_error(3) ..... dbm_open(3)
nextkey(3X) data base subroutines /firstkey(3X) ..... dbminit(3X)
sleep(3) suspend execution for interval ..... sleep(3)
swab(3) swap bytes ..... swab(3)
swab(3) swap bytes ..... swab(3)
with that on disk fsync(2) synchronize a files in-core state ..... fsync(2)
select(2) synchronous I/O multiplexing ..... select(2)
error messages perror(3) sys_errlist(3) sys_nerr(3) system ..... perror(3)
setlogmask(3) control system log syslog(3) openlog(3) closelog(3) ..... syslog(3)
perror(3) sys_errlist(3) sys_nerr(3) system error messages ..... perror(3)
messages psignal(3) sys_siglist(3) system signal ..... psignal(3)
intro(2) introduction to system calls ..... intro(2)
allow signals to interrupt system calls siginterrupt(3) ..... siginterrupt(3)
sys_errlist(3) sys_nerr(3) system error messages perror(3) ..... perror(3)
closelog(3) setlogmask(3) control system log syslog(3) openlog(3) ..... syslog(3)
psignal(3) sys_siglist(3) system signal messages ..... psignal(3)
system(3) issue a shell command ..... system(3)
getdtablesize(2) get descriptor table size ..... getdtablesize(2)
opendir(3) readdir(3) telldir(3) seekdir(3)/ ..... opendir(3)
/tgetstr(3X) tgoto(3X) tputs(3X) terminal independent operation/ ..... tgetent(3X)
stty(3C) gtty(3C) set and get terminal state (defunct) ..... stty(3C)
ttyslot(3) find name of a terminal ttyname(3) isatty(3) ..... ttyname(3)
flushing any pending/ exit(3) terminate a process after ..... exit(3)
tgetflag(3X) tgetstr(3X)/ tgetent(3X) tgetnum(3X) ..... tgetent(3X)
tgetent(3X) tgetnum(3X) tgetflag(3X) tgetstr(3X)/ ..... tgetent(3X)
tgetstr(3X)/ tgetent(3X) tgetnum(3X) tgetflag(3X) ..... tgetent(3X)
/tgetnum(3X) tgetflag(3X) tgetstr(3X) tgoto(3X) tputs(3X)/ ..... tgetent(3X)
/tgetflag(3X) tgetstr(3X) tgoto(3X) tputs(3X) terminal/ ..... tgetent(3X)
time time(3C) ftime(3C) get date and ..... time(3C)
get/set value of interval timer getitimer(2) setitimer(2) ..... getitimer(2)
times(3C) get process times ..... times(3C)
utime(3C) set file times ..... utime(3C)
times(3C) get process times ..... times(3C)
/localtime(3) gmtime(3) asctime(3) timezone(3) convert date and time/ ..... ctime(3)
isascii(3) toupper(3) tolower(3) toascii(3) character/ /iscntrl(3) ..... isalpha(3)
popen(3) pclose(3) initiate I/O to/from a process ..... popen(3)
/iscntrl(3) isascii(3) toupper(3) tolower(3) toascii(3) character/ ..... isalpha(3)
/isgraph(3) iscntrl(3) isascii(3) toupper(3) tolower(3) toascii(3)/ ..... isalpha(3)
operation/ /tgetstr(3X) tgoto(3X) tputs(3X) terminal independent ..... tgetent(3X)
find name of a terminal ttyname(3) isatty(3) ttyslot(3) ..... ttyname(3)
terminal ttyname(3) isatty(3) ttyslot(3) find name of a ..... ttyname(3)

getpw(3C) get name from uid ..... getpw(3C)
into input stream ungetc(3S) push character back ..... ungetc(3S)
mktemp(3) make a unique file name ..... mktemp(3)
pause(3C) stop until signal ..... pause(3C)
setegid(3) setrgid(3) set user and group ID /setgid(3) ..... setuid(3)
set real and effective user IDs setreuid(2) ..... setreuid(2)
endusershell(3) get legal user shells /setusershell(3) ..... getusershell(3)
get information about resource utilization getrusage(2) ..... getrusage(2)
get information about resource utilization vtimes(3C) ..... vtimes(3C)
utime(3C) set file times ..... utime(3C)
allocator valloc(3C) aligned memory ..... valloc(3C)
abs(3) integer absolute value ..... abs(3)
getenv(3) value for environment name ..... getenv(3)
getitimer(2) setitimer(2) get/set value of interval timer ..... getitimer(2)
byte/ /ntohl(3n) ntohs(3n) convert values between host and network ..... htonl(3n)
assert(3) program verification ..... assert(3)
resource utilization vtimes(3C) get information about ..... vtimes(3C)
release blocked signals and wait for interrupt /atomically ..... sigpause(2)
getw(3S) get character or word from stream /fgetc(3S) ..... getc(3S)
putw(3S) put character or word on a stream /fputc(3S) ..... putc(3S)
getwd(3) get current working directory pathname ..... getwd(3)

NAME

intro – introduction to system calls

SYNOPSIS

#include <sys/errno.h>

DESCRIPTION

This section describes some of the 4.3BSD system calls. The calls listed below have been incorporated in Volume 1 (in section 2) of the *Programmer's Reference Manual.*

Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible return value. This is almost always –1; some of the BSD system calls are implemented in the library. Note that a number of system calls overload the meanings of these error numbers, and that the meanings must be interpreted according to the type and circumstances of the call.

acct.2	getpid.2	read.2
brk.2	gettimeofday.2	readlink.2
chdir.2	getuid.2	rename.2
chmod.2	ioctl.2	rmdir.2
chown.2	kill.2	setpgrp.2
chroot.2	link.2	stat.2
close.2	lseek.2	symlink.2
creat.2	mkdir.2	sync.2
execve.2	mknod.2	truncate.2
exit.2	mount.2	umask.2
fcntl.2	open.2	unlink.2
fork.2	pipe.2	wait.2
gethostname.2	profil.2	write.2
getpgrp.2	ptrace.2	

SEE ALSO

intro(3), perror(3)

NAME

accept – accept a connection on a socket

SYNOPSIS

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

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

DESCRIPTION

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

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

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

RETURN VALUE

The call returns –1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

ERRORS

accept fails if:

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

SEE ALSO

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

NAME

access – determine accessibility of file

SYNOPSIS

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

#define R_OK     4    /* test for read permission */
#define W_OK     2    /* test for write permission */
#define X_OK     1    /* test for execute (search) permission */
#define F_OK     0    /* test for presence of file */

accessible = access(path, mode)
int accessible;
char *path;
int mode;
```

DESCRIPTION

access checks the given file *path* for accessibility according to *mode*, which is an inclusive or of the bits R_OK, W_OK, and X_OK. Specifying *mode* as F_OK (i.e., 0) tests whether the directories leading to the file can be searched and the file exists.

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

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

RETURN VALUE

If *path* cannot be found or if any of the desired access modes would not be granted, then a –1 value is returned; otherwise a 0 value is returned.

ERRORS

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

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named file does not exist.
[EACCES]	Search permission is denied for a component of the path prefix.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.
[EROFS]	Write access is requested for a file on a read-only file system.
[ETXTBSY]	Write access is requested for a pure procedure (shared text) file that is being executed.
[EACCES]	Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix. The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits, members of the file's group other than the owner have permission checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits.
[EFAULT]	*path* points outside the process's allocated address space.
[EIO]	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

chmod(2), stat(2)

NAME

bind – bind a name to a socket

SYNOPSIS

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

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

DESCRIPTION

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

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

RETURN VALUE

If the bind is successful, a 0 value is returned. A return value of –1 indicates an error, which is further specified in the global *errno*.

ERRORS

The *bind* call fails if:

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

SEE ALSO

connect(2), listen(2), socket(2), getsockname(2)

NAME

connect – initiate a connection on a socket

SYNOPSIS

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

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

DESCRIPTION

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

RETURN VALUE

If the connection or binding succeeds, then 0 is returned. Otherwise a –1 is returned, and a more specific error code is stored in *errno*.

ERRORS

The *connect* call fails if:

[EBADF]	*s* is not a valid descriptor.
[ENOTSOCK]	*s* is a descriptor for a file, not a socket.
[EADDRNOTAVAIL]	The specified address is not available on this machine.
[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this socket.
[EISCONN]	The socket is already connected.
[ETIMEDOUT]	Connection establishment timed out without establishing a connection.
[ECONNREFUSED]	The attempt to connect was forcefully rejected.
[ENETUNREACH]	The network isn't reachable from this host.
[EADDRINUSE]	The address is already in use.
[EFAULT]	The *name* parameter specifies an area outside the process address space.

SEE ALSO

accept(2), select(2), socket(2), getsockname(2)

NAME

dup, dup2 – duplicate a descriptor

SYNOPSIS

newd = dup(oldd)
int newd, oldd;

dup2(oldd, newd)
int oldd, newd;

DESCRIPTION

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

The object referenced by the descriptor does not distinguish between references using *oldd* and *newd* in any way. Thus if *newd* and *oldd* are duplicate references to an open file, *read*(2), *write*(2) and *lseek*(2) calls all move a single pointer into the file, and append mode, non-blocking I/O and asynchronous I/O options are shared between the references. If a separate pointer into the file is desired, a different object reference to the file must be obtained by issuing an additional *open*(2) call. The close-on-exec flag on the new file descriptor is unset.

In the second form of the call, the value of *newd* desired is specified. If this descriptor is already in use, the descriptor is first deallocated as if a *close*(2) call had been done first.

RETURN VALUE

The value –1 is returned if an error occurs in either call. The external variable *errno* indicates the cause of the error.

ERRORS

Dup and *dup2* fail if:

[EBADF] *Oldd* or *newd* is not a valid active descriptor

[EMFILE] Too many descriptors are active.

SEE ALSO

accept(2), open(2), close(2), fcntl(2), pipe(2), socket(2), socketpair(2), getdtablesize(2)

NAME

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

SYNOPSIS

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

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

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

DESCRIPTION

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

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

The locking mechanism allows two types of locks: *shared* locks and *exclusive* locks. At any time multiple shared locks may be applied to a file, but at no time are multiple exclusive, or both shared and exclusive, locks allowed simultaneously on a file.

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

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

NOTES

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

Processes blocked awaiting a lock may be awakened by signals.

RETURN VALUE

Zero is returned if the operation was successful; on an error a –1 is returned and an error code is left in the global location *errno*.

ERRORS

The *flock* call fails if:

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

SEE ALSO

open(2), close(2), dup(2), execve(2), fork(2)

NAME

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

SYNOPSIS

fsync(fd)
int fd;

DESCRIPTION

fsync causes all modified data and attributes of *fd* to be moved to a permanent storage device. This normally results in all in-core modified copies of buffers for the associated file to be written to a disk.

fsync should be used by programs that require a file to be in a known state, for example, in building a simple transaction facility.

RETURN VALUE

A 0 value is returned on success. A –1 value indicates an error.

ERRORS

The *fsync* fails if:

[EBADF]	*Fd* is not a valid descriptor.
[EIO]	An I/O error occurred while reading from or writing to the file system.

SEE ALSO

sync(2), sync(1M)

NAME

getdtablesize – get descriptor table size

SYNOPSIS

nfds = getdtablesize()
int nfds;

DESCRIPTION

Each process has a fixed size descriptor table, which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The call *getdtablesize* returns the size of this table.

SEE ALSO

close(2), dup(2), open(2), select(2)

NAME

getgid, getegid – get group identity

SYNOPSIS

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

gid = getgid()
gid_t gid;

egid = getegid()
gid_t egid;
```

DESCRIPTION

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

The real group ID is specified at login time.

The effective group ID is more transient, and determines additional access permission during execution of a "set-group-ID" process, and it is for such processes that *getgid* is most useful.

SEE ALSO

getuid(2), setregid(2), setgid(3)

NAME

getitimer, setitimer – get/set value of interval timer

SYNOPSIS

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

#define ITIMER_REAL      0      /* real time intervals */
#define ITIMER_VIRTUAL   1      /* virtual time intervals */
#define ITIMER_PROF      2      /* user and system virtual time */

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

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

DESCRIPTION

The system provides each process with three interval timers, defined in *<sys/time.h>*. The *getitimer* call returns the current value for the timer specified in *which* in the structure at *value*. The *setitimer* call sets a timer to the specified *value* (returning the previous value of the timer if *ovalue* is nonzero).

A timer value is defined by the *itimerval* structure:

```
struct itimerval {
        struct  timeval it_interval;    /* timer interval */
        struct  timeval it_value;       /* current value */
};
```

If *it_value* is non-zero, it indicates the time to the next timer expiration. If *it_interval* is non-zero, it specifies a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 disables a timer. Setting *it_interval* to 0 causes a timer to be disabled after its next expiration (assuming *it_value* is non-zero).

Time values smaller than the resolution of the system clock are rounded up to this resolution (on the VAX, 10 milliseconds).

The ITIMER_REAL timer decrements in real time. A SIGALRM signal is delivered when this timer expires.

The ITIMER_VIRTUAL timer decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.

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

NOTES

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

RETURN VALUE

If the calls succeed, a value of 0 is returned. If an error occurs, the value –1 is returned, and a more precise error code is placed in the global variable *errno*.

ERRORS

The possible errors are:

[EFAULT] The *value* parameter specified a bad address.

[EINVAL] A *value* parameter specified a time was too large to be handled.

SEE ALSO

sigvec(2), gettimeofday(2)

NAME

getrusage – get information about resource utilization

SYNOPSIS

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

#define RUSAGE_SELF      0    /* calling process */
#define RUSAGE_CHILDREN -1    /* terminated child processes */

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

DESCRIPTION

getrusage returns information describing the resources utilized by the current process, or all its terminated child processes. The *who* parameter is one of RUSAGE_SELF or RUSAGE_CHILDREN. The buffer to which *rusage* points is filled in with the following structure:

```
struct rusage {
        struct timeval ru_utime;    /* user time used */
        struct timeval ru_stime;    /* system time used */
        int    ru_maxrss;
        int    ru_ixrss;            /* integral shared text memory size */
        int    ru_idrss;            /* integral unshared data size */
        int    ru_isrss;            /* integral unshared stack size */
        int    ru_minflt;           /* page reclaims */
        int    ru_majflt;           /* page faults */
        int    ru_nswap;            /* swaps */
        int    ru_inblock;          /* block input operations */
        int    ru_oublock;          /* block output operations */
        int    ru_msgsnd;           /* messages sent */
        int    ru_msgrcv;           /* messages received */
        int    ru_nsignals;         /* signals received */
        int    ru_nvcsw;            /* voluntary context switches */
        int    ru_nivcsw;           /* involuntary context switches */
};
```

The fields are interpreted as follows:

ru_utime — the total amount of time spent executing in user mode.

ru_stime — the total amount of time spent in the system executing on behalf of the process(es).

ru_maxrss — the maximum resident set size utilized (in kilobytes).

ru_ixrss — an "integral" value indicating the amount of memory used by the text segment that was also shared among other processes. This value is expressed in units of kilobytes * seconds-of-execution and is calculated by summing the number of shared memory pages in use each time the internal system clock ticks and then averaging over 1 second intervals.

ru_idrss — an integral value of the amount of unshared memory residing in the data segment of a process (expressed in units of kilobytes * seconds-of-execution).

ru_isrss	an integral value of the amount of unshared memory residing in the stack segment of a process (expressed in units of kilobytes * seconds-of-execution).
ru_minflt	the number of page faults serviced without any I/O activity; here I/O activity is avoided by "reclaiming" a page frame from the list of pages awaiting reallocation.
ru_majflt	the number of page faults serviced that required I/O activity.
ru_nswap	the number of times a process was "swapped" out of main memory.
ru_inblock	the number of times the file system had to perform input.
ru_outblock	the number of times the file system had to perform output.
ru_msgsnd	the number of IPC messages sent.
ru_msgrcv	the number of IPC messages received.
ru_nsignals	the number of signals delivered.
ru_nvcsw	the number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).
ru_nivcsw	the number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

NOTES

The numbers *ru_inblock* and *ru_outblock* account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

ERRORS

The possible errors for *getrusage* are:

[EINVAL]	The *who* parameter is not a valid value.
[EFAULT]	The address specified by the *rusage* parameter is not in a valid part of the process address space.

SEE ALSO

gettimeofday(2), wait(2)

BUGS

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

NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

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

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

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

DESCRIPTION

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

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

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

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

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

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

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

SO_ERROR get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a *bind*(2) call should allow reuse of local addresses. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

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

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

RETURN VALUE

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

ERRORS

The call succeeds unless:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOTSOCK]	The argument *s* is a file, not a socket.
[ENOPROTOOPT]	The option is unknown at the level indicated.
[EFAULT]	The address pointed to by *optval* is not in a valid part of the process address space. For *getsockopt*, this error may also be returned if *optlen* is not in a valid part of the process address space.

SEE ALSO

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

BUGS

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

NAME

killpg – send signal to a process group

SYNOPSIS

killpg(pgrp, sig)
int pgrp, sig;

DESCRIPTION

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

The sending process and members of the process group must have the same effective user ID, or the sender must be the super-user. As a single special case the continue signal SIGCONT may be sent to any process that is a descendant of the current process.

RETURN VALUE

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

ERRORS

killpg fails and no signal is sent if any of the following occur:

[EINVAL]	*sig* is not a valid signal number.
[ESRCH]	No process can be found in the process group specified by *pgrp*.
[ESRCH]	The process group was given as 0 but the sending process does not have a process group.
[EPERM]	The sending process is not the super-user and one or more of the target processes has an effective user ID different from that of the sending process.

SEE ALSO

kill(2), getpgrp(2), sigvec(2)

NAME

listen – listen for connections on a socket

SYNOPSIS

listen(s, backlog)
int s, backlog;

DESCRIPTION

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

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

RETURN VALUE

A 0 return value indicates success; –1 indicates an error.

ERRORS

The call fails if:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOTSOCK]	The argument *s* is not a socket.
[EOPNOTSUPP]	The socket is not of a type that supports the operation *listen*.

SEE ALSO

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

BUGS

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

NAME

recv, recvfrom – receive a message from a socket

SYNOPSIS

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

cc = recv(s, buf, len, flags)
int cc, s;
char *buf;
int len, flags;

cc = recvfrom(s, buf, len, flags, from, fromlen)
int cc, s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;
```

DESCRIPTION

recv and *recvfrom* are used to receive messages from a socket.

The *recv* call is normally used only on a *connected* socket (see *connect*(2)), while *recvfrom* may be used to receive data on a socket whether it is in a connected state or not.

If *from* is non-zero, the source address of the message is filled in. *fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned in *cc*. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see *socket*(2)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see *ioctl*(2)) in which case a *cc* of –1 is returned with the external variable errno set to EWOULDBLOCK.

The *select*(2) call may be used to determine when more data arrives.

The *flags* argument to a recv call is formed by *or*'ing one or more of the values,

```
#define MSG_OOB         0x1     /* process out-of-band data */
```

RETURN VALUE

These calls return the number of bytes received, or –1 if an error occurred.

ERRORS

The calls fail if:

[EBADF]	The argument *s* is an invalid descriptor.
[ENOTSOCK]	The argument *s* is not a socket.
[EWOULDBLOCK]	The socket is marked non-blocking and the receive operation would block.
[EINTR]	The receive was interrupted by delivery of a signal before any data was available for the receive.
[EFAULT]	The data was specified to be received into a non-existent or protected part of the process address space.

SEE ALSO

fcntl(2), read(2), send(2), select(2), getsockopt(2), socket(2)

NAME

select – synchronous I/O multiplexing

SYNOPSIS

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

nfound = select(nfds, readfds, writefds, exceptfds, timeout)
int nfound, nfds;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET(fd, &fdset)
FD_CLR(fd, &fdset)
FD_ISSET(fd, &fdset)
FD_ZERO(&fdset)
int fd;
fd_set fdset;
```

DESCRIPTION

select examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first *nfds* descriptors are checked in each set; i.e. the descriptors from 0 through *nfds*–1 in the descriptor sets are examined. On return, *select* replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned in *nfound*.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: *FD_ZERO(&fdset)* initializes a descriptor set *fdset* to the null set. *FD_SET(fd, &fdset)* includes a particular descriptor *fd* in *fdset*. *FD_CLR(fd, &fdset)* removes *fd* from *fdset*. *FD_ISSET(fd, &fdset)* is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to *FD_SETSIZE*, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is a non-zero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the select blocks indefinitely. To affect a poll, the *timeout* argument should be non-zero, pointing to a zero-valued timeval structure.

Any of *readfds*, *writefds*, and *exceptfds* may be given as zero pointers if no descriptors are of interest.

RETURN VALUE

select returns the number of ready descriptors that are contained in the descriptor sets, or –1 if an error occurred. If the time limit expires then *select* returns 0. If *select* returns with an error, including one due to an interrupted call, the descriptor sets are not modified.

ERRORS

An error return from *select* indicates:

[EBADF]	One of the descriptor sets specified an invalid descriptor.
[EINTR]	A signal was delivered before the time limit expired and before any of the selected events occurred.

[EINVAL] The specified time limit is invalid. One of its components is negative or too large.

SEE ALSO

accept(2), connect(2), read(2), write(2), recv(2), send(2), getdtablesize(2)

BUGS

Although the provision of *getdtablesize*(2) was intended to allow user programs to be written independent of the kernel limit on the number of open files, the dimension of a sufficiently large bit field for select remains a problem. The default size FD_SETSIZE (currently 256) is somewhat larger than the current kernel limit to the number of open files. However, in order to accommodate programs which might potentially use a larger number of open files with select, it is possible to increase this size within a program by providing a larger definition of FD_SETSIZE before the inclusion of <sys/types.h>.

select should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout value is not modified by the *select* call.

NAME

send, sendto – send a message from a socket

SYNOPSIS

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

cc = send(s, msg, len, flags)
int cc, s;
char *msg;
int len, flags;

cc = sendto(s, msg, len, flags, to, tolen)
int cc, s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;
```

DESCRIPTION

send and *sendto* are used to transmit a message to another socket. *send* may be used only when the socket is in a *connected* state, while *sendto* may be used at any time.

The address of the target is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a *send*. Return values of –1 indicate some locally detected errors.

If no messages space is available at the socket to hold the message to be transmitted, then *send* normally blocks, unless the socket has been placed in non-blocking I/O mode. The *select*(2) call may be used to determine when it is possible to send more data.

The *flags* parameter may include one or more of the following:

```
#define MSG_OOB          0x1    /* process out-of-band data */
```

The flag MSG_OOB is used to send "out-of-band" data on sockets that support this notion (e.g. SOCK_STREAM); the underlying protocol must also support "out-of-band" data.

RETURN VALUE

The call returns the number of characters sent, or –1 if an error occurred.

ERRORS

[EBADF]	An invalid descriptor was specified.
[ENOTSOCK]	The argument *s* is not a socket.
[EFAULT]	An invalid user space address was specified for a parameter.
[EMSGSIZE]	The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.
[EWOULDBLOCK]	The socket is marked non-blocking and the requested operation would block.
[ENOBUFS]	The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.

[ENOBUFS] The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion.

SEE ALSO

fcntl(2), recv(2), select(2), getsockopt(2), socket(2), write(2)

NAME

setregid – set real and effective group ID

SYNOPSIS

setregid(rgid, egid)
int rgid, egid;

DESCRIPTION

The real and effective group ID's of the current process are set to the arguments. Unprivileged users may change the real group ID to the effective group ID and vice-versa; only the super-user may make other changes.

Supplying a value of –1 for either the real or effective group ID forces the system to substitute the current ID in place of the –1 parameter.

RETURN VALUE

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

ERRORS

[EPERM] The current process is not the super-user and a change other than changing the effective group-id to the real group-id was specified.

SEE ALSO

getgid(2), setreuid(2), setgid(3)

NAME

setreuid – set real and effective user ID's

SYNOPSIS

setreuid(ruid, euid)
int ruid, euid;

DESCRIPTION

The real and effective user ID's of the current process are set according to the arguments. If *ruid* or *euid* is –1, the current uid is filled in by the system. Unprivileged users may change the real user ID to the effective user ID and vice-versa; only the super-user may make other changes.

RETURN VALUE

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

ERRORS

[EPERM] The current process is not the super-user and a change other than changing the effective user-id to the real user-id was specified.

SEE ALSO

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

NAME

shutdown – shut down part of a full-duplex connection

SYNOPSIS

shutdown(s, how)
int s, how;

DESCRIPTION

The *shutdown* call causes all or part of a full-duplex connection on the socket associated with *s* to be shut down. If *how* is 0, then further receives are disallowed. If *how* is 1, then further sends are disallowed. If *how* is 2, then further sends and receives are disallowed.

DIAGNOSTICS

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

ERRORS

The call succeeds unless:

[EBADF]	*s* is not a valid descriptor.
[ENOTSOCK]	*s* is a file, not a socket.
[ENOTCONN]	The specified socket is not connected.

SEE ALSO

connect(2), socket(2)

NAME

sigblock – block signals

SYNOPSIS

```
#include <signal.h>

sigblock(mask);
int mask;

mask = sigmask(signum)
```

DESCRIPTION

sigblock causes the signals specified in *mask* to be added to the set of signals currently being blocked from delivery. Signals are blocked if the corresponding bit in *mask* is a 1; the macro *sigmask* is provided to construct the mask for a given *signum*.

It is not possible to block SIGKILL, SIGSTOP, or SIGCONT; this restriction is silently imposed by the system.

RETURN VALUE

The previous set of masked signals is returned.

SEE ALSO

kill(2), sigvec(2), sigsetmask(2)

NAME

sigpause – atomically release blocked signals and wait for interrupt

SYNOPSIS

sigpause(sigmask)
int sigmask;

DESCRIPTION

sigpause assigns *sigmask* to the set of masked signals and then waits for a signal to arrive; on return the set of masked signals is restored. *sigmask* is usually 0 to indicate that no signals are now to be blocked. *sigpause* always terminates by being interrupted, returning –1 with *errno* set to EINTR.

In normal usage, a signal is blocked using *sigblock*(2), to begin a critical section, variables modified on the occurrence of the signal are examined to determine that there is no work to be done, and the process pauses awaiting work by using *sigpause* with the mask returned by *sigblock*.

SEE ALSO

sigblock(2), sigvec(2)

NAME

sigreturn – return from signal

SYNOPSIS

```
#include <signal.h>
sigreturn(scp);
struct sigcontext *scp;
```

DESCRIPTION

sigreturn allows users to atomically unmask, switch stacks, and return from a signal context. The processes signal mask and stack status are restored from the context. The system call does not return; the users stack pointer, frame pointer, argument pointer, and processor status longword are restored from the context. Execution resumes at the specified pc. This system call is used by the trampoline code, and *longjmp*(3) when returning from a signal to the previously executing program.

NOTES

This system call is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUE

If successful, the system call does not return. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

ERRORS

sigreturn fails and the process context does not change if one of the following occurs.

[EFAULT] *scp* points to memory that is not a valid part of the process address space.

[EINVAL] The process status longword is invalid or would improperly raise the privilege level of the process.

SEE ALSO

sigvec(2), setjmp(3)

NAME

sigsetmask – set current signal mask

SYNOPSIS

#include <signal.h>

sigsetmask(mask);
int mask;

mask = sigmask(signum)

DESCRIPTION

sigsetmask sets the current signal mask (those signals that are blocked from delivery). Signals are blocked if the corresponding bit in *mask* is a 1; the macro *sigmask* is provided to construct the mask for a given *signum*.

The system quietly disallows SIGKILL, SIGSTOP, or SIGCONT to be blocked.

RETURN VALUE

The previous set of masked signals is returned.

SEE ALSO

kill(2), sigvec(2), sigblock(2), sigpause(2)

NAME

sigstack – set and/or get signal stack context

SYNOPSIS

```
#include <signal.h>

struct sigstack {
        caddr_t ss_sp;
        int     ss_onstack;
};

sigstack(ss, oss);
struct sigstack *ss, *oss;
```

DESCRIPTION

sigstack allows users to define an alternate stack on which signals are to be processed. If *ss* is non-zero, it specifies a *signal stack* on which to deliver signals and tells the system if the process is currently executing on that stack. When a signal's action indicates its handler should execute on the signal stack (specified with a *sigvec*(2) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler's execution. If *oss* is non-zero, the current signal stack state is returned.

NOTES

Signal stacks are not grown automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

RETURN VALUE

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

ERRORS

sigstack fails and the signal stack context does not change if the following occurs:

[EFAULT] Either *ss* or *oss* points to memory that is not a valid part of the process address space.

SEE ALSO

sigvec(2), setjmp(3)

NAME

sigvec – software signal facilities

SYNOPSIS

```
#include <signal.h>

struct sigvec {
        int     (*sv_handler)();
        int     sv_mask;
        int     sv_flags;
};

sigvec(sig, vec, ovec)
int sig;
struct sigvec *vec, *ovec;
```

DESCRIPTION

The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored*. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special *signal stack*.

All signals have the same *priority*. Signal routines execute with the signal that caused their invocation *blocked*, but other signals may yet occur. A global *signal mask* defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a *sigblock*(2) or *sigsetmask*(2) call, or when a signal is delivered to the process.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a *sigblock* or *sigsetmask* call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and *or*'ing in the signal mask associated with the handler to be invoked.

sigvec assigns a handler for a specific signal. If *vec* is non-zero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if the SV_ONSTACK bit is set in *sv_flags*, the system will deliver the signal to the process on a *signal stack*, specified with *sigstack*(2). If *ovec* is non-zero, the previous handling information for the signal is returned to the user.

The following list gives all signals with names as they appear in the include file *<signal.h>*:

SIGHUP	1	hangup
SIGINT	2	interrupt (rubout)
SIGQUIT	3*	quit (ASCII FS)
SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)

SIGIOT	6*	IOT instruction (obsolete)
SIGABRT	6*	used by abort, replaces SIGIOT
SIGEMT	7*	EMT instruction (obsolete)
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user-defined signal 1
SIGUSR2	17	user-defined signal 2
SIGCLD	18	• death of a child
SIGPWR	19	power-fail restart
SIGWIND	20	• window change
SIGURG	21	• urgent condition on an I/O channel
SIGIO	22	• pollable event occured
SIGSTOP	23	† sendable stop signal, not from tty
SIGTSTP	24	† stop signal from tty
SIGTTIN	25	† process stop by background tty read
SIGTTOU	26	† process stop by background tty write
SIGCONT	27	• continue a stopped process
SIGXCPU	28	exceeded CPU time limit
SIGXFSZ	29	exceeded file size limit
SIGVTALRM	30	virtual time alarm
SIGPROF	31	profiling time alarm

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another *sigvec* call is made, or an *execve*(2) is performed. The default action for a signal may be reinstated by setting *sv_handler* to SIG_DFL; 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 *sv_handler* is SIG_IGN the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain system calls, the call is normally restarted. The call can be forced to terminate prematurely with an EINTR error return by setting the SV_INTERRUPT bit in *sv_flags*. The affected system calls are *read*(2) or *write*(2) on a slow device (such as a terminal; but not a file) and during a *wait*(2).

After a *fork*(2) or *vfork*(2) the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt flags.

execve(2) resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt system calls continue to do so.

NOTES

The mask specified in *vec* is not allowed to block SIGKILL, SIGSTOP, or SIGCONT. This is done silently by the system.

The handler routine to catch signals can be delared as follows:

```
handler(sig, code, junk, context)
int sig, code, junk
struct sigcontext *context;
```

Here, *sig* is the signal number. *code* is a value which further interprets *sig*; it may be one of the following:

SIGFPE
0: integer overflow
1: floating exception

SIGTRAP
CAUSESINGLE: single step
CAUSEBREAK: breakpoint instruction

The value of *junk* is the address of the handler routine itself. *context* is a pointer to the machine state at the time of the exception. It is defined in */usr/include/sys/signal.h*.

On floating point exceptions, the FPU is halted. Explicit user action of restarting the FPU will be necessary.

RETURN VALUE

A 0 value indicated that the call succeeded. A –1 return value indicates an error occurred and *errno* is set to indicated the reason.

ERRORS

sigvec fails and no new signal handler is installed if one of the following occurs:

[EFAULT]	Either *vec* or *ovec* points to memory that is not a valid part of the process address space.
[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), sigblock(2), sigsetmask(2), sigpause(2), sigstack(2), sigvec(2), setjmp(3), siginterrupt(3), tty(7)

NAME

socket – create an endpoint for communication

SYNOPSIS

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

s = socket(domain, type, protocol)
int s, domain, type, protocol;

DESCRIPTION

socket creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file *<sys/socket.h>*. The currently understood formats are

PF_UNIX	(UNIX internal protocols),
PF_INET	(ARPA Internet protocols),

The socket has the indicated *type,* which specifies the semantics of communication. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). SOCK_RAW sockets provide access to internal network protocols and interfaces. The type SOCK_RAW is available only to the super-user.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see *protocols*(3N).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a *connect*(2) call. Once connected, data may be transferred using *read*(2) and *write*(2) calls or some variant of the *send*(2) and *recv*(2) calls. When a session has been completed a *close*(2) may be performed. Out-of-band data may also be transmitted as described in *send*(2) and received as described in *recv*(2).

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls indicate an error with –1 returns and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in *send*(2) calls. Datagrams are generally received with *recvfrom*(2), which returns the next datagram with its return address.

An *fcntl*(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives.

RETURN VALUE

A –1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

ERRORS

The *socket* call fails if:

[EPROTONOSUPPORT] The protocol type or the specified protocol is not supported within this domain.

[EMFILE] The per-process descriptor table is full.

[ENFILE] The system file table is full.

[EACCESS] Permission to create a socket of the specified type and/or protocol is denied.

[ENOBUFS] Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

SEE ALSO

accept(2), bind(2), connect(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), read(2), recv(2), select(2), send(2), shutdown(2), write(2)

NAME

intro – introduction to C library functions

DESCRIPTION

This section describes functions that may be found in the 4.3BSD libraries. The library functions are those other than the functions which directly invoke operating 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). The link editor *ld*(1) searches this library when the **–43** option is invoked. The C library also includes all the functions described in section 2.

Library functions available from FORTRAN are described separately in the *Fortran Reference Manual*.

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

(3) These functions are the standard C library functions. They correspond to the System V Release 3.0 3C 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, which constitute the math library *libm*, are listed in Volume 1 of the *Programmer's Reference Manual* . They no longer appear in this manual.

(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
/lib/bsd/libc.a	the 4.3 BSD 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
/usr/lib/bsd/libm_p.a	the 4.3 BSD math library compiled for profiling

SEE ALSO

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

NAME

abort – generate a fault

DESCRIPTION

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

SEE ALSO

exit(2), sigvec(2)

DIAGNOSTICS

Usually "Illegal instruction – core dumped" from the shell.

BUGS

abort 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)

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

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)
int i;
```

DESCRIPTION

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

bcopy copies *length* bytes from string *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 finds 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 that the value passed is zero.

BUGS

The *bcopy* routine take parameters backwards from *strcpy*.

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 (DES), 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*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

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 – convert date and time to ASCII

SYNOPSIS

```
char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

ctime converts a time 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\0

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 */
};
```

When local time is called for, the program consults the system to determine the time zone and whether the U.S.A., Australian, Eastern European, Middle European, or Western European daylight saving time adjustment is appropriate. The program knows about various peculiarities in time conversion over the past 10-20 years; if necessary, this understanding can be extended.

timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; 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 produced.

SEE ALSO

gettimeofday(2), time(3)

BUGS

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

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(5)

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;

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.*

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

close(2), dir(4), lseek(2), open(2), read(2)

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 I/O (*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

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.

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*(5) 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

csh(1) environ(5), execve(2), fork(2)

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*(4)), 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]* are 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

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* = $x * 2^n$ indirectly through *eptr.*

ldexp returns the quantity *value* $* 2^{exp}$.

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

NAME

getenv – value for environment name

SYNOPSIS

char *getenv(name)
char *name;

DESCRIPTION

getenv searches the environment list (see *environ*(5)) for a string of the form *name=value* and returns a pointer to the string *value* if such a string is present, otherwise *getenv* returns the value 0 (NULL).

SEE ALSO

environ(5), execve(2)

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.

```
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(4)

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

getlogin – get login name

SYNOPSIS

char *getlogin()

DESCRIPTION

getlogin returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same userid is shared by several login names.

If *getlogin* is called within a process that is not attached to a terminal, or if there is no entry in */etc/utmp* for the process's terminal, *getlogin* returns NULL. A reasonable procedure for determining the login name is to first call *getlogin* and if it fails, to call *getpwuid*(*getuid*()).

FILES

/etc/utmp

SEE ALSO

getpwent(3), utmp(4), ttyslot(3)

DIAGNOSTICS

Returns NULL if name not found.

BUGS

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

NAME

getopt – get option letter from argv

SYNOPSIS

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

extern char *optarg;
extern int optind;
```

DESCRIPTION

getopt returns the next option letter in *argv* that matches a letter in *optstring*. *optstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. *optarg* is set to point to the start of the option argument on return from *getopt*.

getopt places in *optind* the *argv* index of the next argument to be processed. Because *optind* is external, it is normally initialized to zero automatically before the first call to *getopt*.

When all options have been processed (i.e., up to the first non-option argument), *getopt* returns EOF. The special option -- may be used to delimit the end of the options; EOF is returned, and -- skipped.

DIAGNOSTICS

getopt prints an error message on *stderr* and returns a question mark (?) when it encounters an option letter not included in *optstring*.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the options **f** and **o**, both of which require arguments:

```
main(argc, argv)
int argc;
char **argv;
{
        int c;
        extern int optind;
        extern char *optarg;
        .
        .
        .
        while ((c = getopt(argc, argv, "abf:o:")) != EOF)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc();
```

```
                        break;
                case 'f':
                        ifile = optarg;
                        break;
                case 'o':
                        ofile = optarg;
                        break;
                case '?':
                default:
                        errflg++;
                        break;
                }
        if (errflg) {
                fprintf(stderr, "Usage: ...");
                exit(2);
        }
        for (; optind < argc; optind++) {
                .
                .
                .
        }
        .
        .
        .
}
```

HISTORY

Written by Henry Spencer, working from a Bell Labs manual page. Modified by Keith Bostic to behave more like the System V version.

BUGS

It is not obvious how '–' standing alone should be treated; this version treats it as a non-option argument, which is not always right.

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

getopt is quite flexible but the obvious price must be paid: there is much it could do that it doesn't, like checking mutually exclusive options, checking type of option arguments, etc.

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

getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

struct passwd *getpwent()

setpwent()

endpwent()

setpwfile(name)
char *name;

DESCRIPTION

getpwent, *getpwuid* and *getpwnam* each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct   passwd { /* see getpwent(3) */
         char    *pw_name;
         char    *pw_passwd;
         int     pw_uid;
         int     pw_gid;
         int     pw_quota;
         char    *pw_comment;
         char    *pw_gecos;
         char    *pw_dir;
         char    *pw_shell;
};

         struct passwd *getpwent(), *getpwuid(), *getpwnam();
```

The fields *pw_quota* and *pw_comment* are unused; the others have meanings described in *passwd*(4).

Searching the password file is done using the *ndbm* database access routines. *setpwent* opens the database; *endpwent* closes it. *getpwuid* and *getpwnam* search the database (opening it if necessary) for a matching *uid* or *name*. EOF is returned if there is no entry.

For programs wishing to read the entire database, *getpwent* reads the next line (opening the database if necessary). In addition to opening the database, *setpwent* can be used to make *getpwent* begin its search from the beginning of the database.

setpwfile changes the default password file to *name* thus allowing alternate password files to be used. Note that it does *not* close the previous file. If this is desired, *endpwent* should be called prior to it.

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(4)

DIAGNOSTICS

The routines *getpwent*, *getpwuid*, and *getpwnam*, return NULL on EOF or error.

BUGS

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

NAME

getusershell, setusershell, endusershell – get legal user shells

SYNOPSIS

char *getusershell()

setusershell()

endusershell()

DESCRIPTION

getusershell returns a pointer to a legal user shell as defined by the system manager in the file */etc/shells*. If */etc/shells* does not exist, the two standard system shells */bin/sh* and */bin/csh* are returned.

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

FILES

/etc/shells

DIAGNOSTICS

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

BUGS

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

NAME

getwd – get current working directory pathname

SYNOPSIS

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

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.

NAME

malloc, free, realloc, calloc – 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;
```

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 results 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 are not changed 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.

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 is 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.

NAME

mktemp – make a unique file name

SYNOPSIS

char *mktemp(template)
char *template;

mkstemp(template)
char *template;

DESCRIPTION

mktemp creates a unique file name, typically in a temporary filesystem, by replacing *template* with a unique file name, and returns the address of the template. The template should contain a file name with six trailing X's, which are replaced with the current process id and a unique letter. *mkstemp* makes the same replacement to the template but returns a file descriptor for the template file open for reading and writing. *mkstemp* avoids the race between testing whether the file exists and opening it for use.

SEE ALSO

getpid(2), open(2)

DIAGNOSTICS

mkstemp returns an open file descriptor upon success. It 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*(2) buffer allocation. An executable program created by:

cc –pg . . .

automatically includes calls for the *gprof*(1) monitor.

monstartup is a high level interface to *profil*. *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 is 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), gprof(1), prof(1), profil(2), sbrk(2)

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 handle very large (a billion blocks) databases and 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.

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

Before a database can be accessed, it must be opened by *dbm_open*. This opens or creates 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 only inserts new entries into the database and does not change an existing entry with the same key. DBM_REPLACE replaces 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* returns the first key in the database. *dbm_nextkey* returns the next key in the database. This code traverses 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 contains 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* returns 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

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*(4) for the structure declaration.

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

SEE ALSO

a.out(4)

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

perror, sys_errlist, sys_nerr – system error messages

SYNOPSIS

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

DESCRIPTION

perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno* (see *intro*(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the newline. *sys_nerr* is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2), psignal(3)

NAME

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

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

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

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. NSIG defined in *<signal.h>* is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO

perror(3), sigvec(2)

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

random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS

long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;

DESCRIPTION

random uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16\times(2^{31}-1)$.

Random and *srandom* have (almost) the same calling sequence and initialization properties as *rand* and *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" produces 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 or changing random number generators). Like *rand*(3), however, *random* produces by default 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. (Current "optimal" values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts are rounded down to the nearest known amount. Using less than 8 bytes causes 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 two-thirds 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*(1M) 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 non-zero, then an auxiliary channel to a control process is set up, and a descriptor for it is placed in **fd2p*. The control process returns diagnostic output from the command (unit 2) on this channel, and also accepts 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) is 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*(1M).

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

intro(2), rexec(3), rexecd(1M), rlogin(1C), rlogind(1M), rsh(1C), rshd(1M)

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), egrep(1), ex(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 to 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* as follows. Options are a simple bit mask and are OR'ed in to enable.

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. *res_send* continues until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
: Use TCP connections for queries instead of UDP.

RES_STAYOPEN
: Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC
Unused currently (ignore truncation errors, 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
Append the default domain name to single label queries. This is the default.

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* returns the size of the query or –1 if the query is larger than *buflen*. *op* is usually QUERY but can be any of the query types defined in *nameser.h*. *dname* is the domain name. If *dname* consists of a single label and the RES_DEFNAMES flag is enabled (the default), *dname* is appended to the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable LOCALDOMAIN. *newrr* is currently unused but is intended for making update messages.

res_send sends a query to name servers and returns an answer. It calls *res_init* if RES_INIT is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or –1 if there were errors.

dn_expand expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or –1 if there was an error.

dn_comp compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or –1 if there were errors. *length is the size of the 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, we don't try to compress names. If *lastdnptr* is NULL, we don't update the list.

FILES

/etc/resolv.conf see resolver(4)

SEE ALSO

named(1M), resolver($)

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)) returns a pointer to a structure, which contains the necessary port. The protocol for connection is described in detail in *rexecd*(1M).

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 is setup, and a descriptor for it placed in **fd2p*. The control process returns diagnostic output from the command (unit 2) on this channel, and also accepts 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) is 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(1M)

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 are 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

dir(4), directory(3), malloc(3), qsort(3)

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

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

signal(3), sigstack(2), sigvec(2)

NAME

setuid, seteuid, setruid, setgid, setegid, setrgid – set user and group ID

SYNOPSIS

#include <sys/types.h>

setuid(uid)
seteuid(euid)
setruid(ruid)
uid_t uid, euid, ruid;

setgid(gid)
setegid(egid)
setrgid(rgid)
gid_t gid, egid, rgid;

DESCRIPTION

setuid (*setgid*) sets both the real and effective user ID (group ID) of the current process as specified.

seteuid (*setegid*) sets the effective user ID (group ID) of the current process.

setruid (*setrgid*) sets the real user ID (group ID) of the current process.

These calls are only permitted to the super-user or if the argument is the real or effective ID.

SEE ALSO

getgid(2), getuid(2), setregid(2), setreuid(2)

DIAGNOSTICS

Zero is returned if the user (group) ID is set; –1 is returned otherwise.

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 are restarted if they are interrupted by the specified signal and no data has been transferred yet. System call restart is the default behavior on 4.2BSD.

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 returns –1 with errno set to EINTR. Interrupted system calls that have started transferring data returns the amount of data actually transferred. System call interrupt is the signal behavior found on 4.1BSD and AT&T System V UNIX systems.

Note that the new 4.2BSD signal handling semantics are not altered in any other way. Most notably, signal handlers always remain installed until explicitly changed by a subsequent *sigvec*(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 causes 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. 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

sigblock(2), sigpause(2), sigsetmask(2), sigvec(2)

NAME

sleep – suspend execution for interval

SYNOPSIS

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

setitimer(2), sigpause(2)

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex – string operations

SYNOPSIS

#include <strings.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

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 *s2* to the end of string *s1*. *strncat* copies at most *n* 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 *n* characters.

strcpy copies string *s2* to *s1*, stopping after the null character has been moved. *strncpy* copies exactly *n* characters, truncating or null-padding *s2;* the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*.

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.

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 PDP-11's and other machines. *nbytes* should be even.

NAME

syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logopt, facility)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

setlogmask(maskpri)

DESCRIPTION

syslog arranges to write *message* onto the system log maintained by *syslogd*(1M). The message is tagged with *priority*. The message looks like a *printf*(3) string except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message is read by *syslogd*(1M) and written to the system console, log files, or forwarded to *syslogd* on another host as appropriate.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from an ordered list:

LOG_EMERG	A panic condition. This is normally broadcast to all users.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions, 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.

If *syslog* cannot pass the message to *syslogd*, it attempts to write the message on */dev/console* if the LOG_CONS option is set (see below).

If special processing is needed, *openlog* can be called to initialize the log file. The parameter *ident* is a string that is prepended to every message. *logopt* is a bit field indicating logging options. Current values for *logopt* are:

LOG_PID	Log the process id with each message: useful for identifying instantiations of daemons.
LOG_CONS	Force writing messages to the console if unable to send it to *syslogd*. This option is safe to use in daemon processes that have no controlling terminal since *syslog* forks before opening the console.
LOG_NDELAY	Open the connection to *syslogd* immediately. Normally the open is delayed until the first message is logged. Useful for programs that need to manage the order in which file descriptors are allocated.

LOG_NOWAIT	Don't wait for children forked to log messages on the console. This option should be used by processes that enable notification of child termination via SIGCHLD, as *syslog* may otherwise block waiting for a child whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility encoded:

LOG_KERN	Messages generated by the kernel. These cannot be generated by any user processes.
LOG_USER	Messages generated by random user processes. This is the default facility identifier if none is specified.
LOG_MAIL	The mail system.
LOG_DAEMON	System daemons, such as *ftpd*(1M), *routed*(1M), etc.
LOG_AUTH	The authorization system: *login*(1), *su*(1), *getty*(1M), etc.
LOG_LPR	The line printer spooling system: *lpr*(1), *lpc*(1M), *lpd*(1M), etc.
LOG_LOCAL0	Reserved for local use. Similarly for LOG_LOCAL1 through LOG_LOCAL7.

closelog can be used to close the log file.

setlogmask sets the log priority mask to *maskpri* and returns the previous mask. Calls to *syslog* with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro LOG_MASK(*pri*); the mask for all priorities up to and including *toppri* is given by the macro LOG_UPTO(*toppri*). The default allows all priorities to be logged.

EXAMPLES

```
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(1M)

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

execve(2), popen(3S), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

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*(4) file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(4)

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

alarm – schedule signal after specified time

SYNOPSIS

alarm(seconds)
unsigned seconds;

DESCRIPTION

This interface is superseded 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

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

NAME

getpw – get name from uid

SYNOPSIS

getpw(uid, buf)
char *buf;

DESCRIPTION

getpw is superseded by *getpwuid*(3).

getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3), passwd(4)

DIAGNOSTICS

Non-zero return on error.

NAME

nice – set program priority

SYNOPSIS

nice(incr)

DESCRIPTION

This interface is superseded 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

fork(2), nice(1), renice(1M), setpriority(2)

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 returns.

RETURN VALUE

Always returns –1.

ERRORS

pause always returns:

[EINTR] The call was interrupted.

SEE ALSO

kill(2), select(2), sigpause(2)

NAME

rand, srand – random number generator

SYNOPSIS

void srand(seed)
unsigned 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 $2^{15}-1$.

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

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 list of all signals with names is given in the include file *<signal.h>*:

IGHUP	1	hangup
SIGINT	2	interrupt (rubout)
SIGQUIT	3 *	quit (ASCII FS)
SIGILL	4 *	illegal instruction (not reset when caught)
SIGTRAP	5 *	trace trap (not reset when caught)
SIGIOT	6 *	IOT instruction (obsolete)
SIGABRT	6 *	used by abort, replaces SIGIOT
SIGEMT	7 *	EMT instruction (obsolete)
SIGFPE	8 *	floating point exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10 *	bus error
SIGSEGV	11 *	segmentation violation
SIGSYS	12 *	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user-defined signal 1
SIGUSR2	17	user-defined signal 2
SIGCLD	18 •	death of a child
SIGPWR	19	power-fail restart
SIGWIND	20 •	window change
SIGURG	21 •	urgent condition on an I/O channel
SIGIO	22 •	pollable event occured
SIGSTOP	23 †	sendable stop signal, not from tty
SIGTSTP	24 †	stop signal from tty
SIGTTIN	25 †	process stop by background tty read
SIGTTOU	26 †	process stop by background tty write
SIGCONT	27 •	continue a stopped process
SIGXCPU	28	exceeded CPU time limit
SIGXFSZ	29	exceeded file size limit
SIGVTALRM	30	virtual time alarm
SIGPROF	31	profiling time alarm

The starred signals generate 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 fails and no action takes 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), kill(2), ptrace(2), setjmp(3), sigblock(2), sigpause(2), sigsetmask(2), sigstack(2), sigvec(2), tty(7)

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 superseded 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*(7) 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(7)

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 superseded 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>*:

```
/*
 * 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

ctime(3), date(1), gettimeofday(2), settimeofday(2)

NAME

times – get process times

SYNOPSIS

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

times(buffer)
struct tms *buffer;
```

DESCRIPTION

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*:

```
/*
 * 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

getrusage(2), time(1), time(3), wait3(2)

NAME

utime – set file times

SYNOPSIS

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

utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

This interface is superseded 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

stat(2), utimes(2)

NAME

valloc – aligned memory allocator

SYNOPSIS

char *valloc(size)
unsigned size;

DESCRIPTION

valloc is superseded by the current version of *malloc*(3), which aligns page-sized and larger allocations.

valloc allocates *size* bytes aligned on a page boundary. It is implemented by calling *malloc* 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

vtimes – get information about resource utilization

SYNOPSIS

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;

DESCRIPTION

This facility is superseded by getrusage(2).

vtimes returns accounting information for the current process and for the terminated child processes of the current process. Either *par_vm* or *ch_vm* or both may be 0, in which case only the information for the pointers which are non-zero is returned.

```
struct vtimes {
        int     vm_utime;               /* user time (*HZ) */
        int     vm_stime;               /* system time (*HZ) */
        /* divide next two by utime+stime to get averages */
        unsigned vm_idsrss;             /* integral of d+s rss */
        unsigned vm_ixrss;              /* integral of text rss */
        int     vm_maxrss;              /* maximum rss */
        int     vm_majflt;              /* major page faults */
        int     vm_minflt;              /* minor page faults */
        int     vm_nswap;               /* number of swaps */
        int     vm_inblk;               /* block reads */
        int     vm_oublk;               /* block writes */
};
```

The *vm_utime* and *vm_stime* fields give the user and system time respectively in 60ths of a second (or 50ths if that is the frequency of wall current in your locality.) The *vm_idrss* and *vm_ixrss* measure memory usage. They are computed by integrating the number of memory pages in use over CPU time. They are reported as though computed discretely, adding the current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core pages over 1 CPU-second for its data and stack, then *vm_idsrss* would have the value 5*60, where *vm_utime+vm_stime* would be the 60. *vm_idsrss* integrates data and stack segment usage, while *vm_ixrss* integrates text segment usage. *vm_maxrss* reports the maximum instantaneous sum of the text+data+stack core-resident page count.

The *vm_majflt* field gives the number of page faults which resulted in disk activity; the *vm_minflt* field gives the number of page faults incurred in simulation of reference bits; *vm_nswap* is the number of swaps which occurred. The number of file system I/O events are reported in *vm_inblk* and *vm_oublk* These numbers account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

SEE ALSO

getrusage(2), time(2), wait3(2)

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

gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent – 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()
```

DESCRIPTION

gethostbyname and *gethostbyaddr* each returns a pointer to an object with the following structure. 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.

sethostent allows a request for the use of a connected socket using TCP 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 *gethostbyname* or *gethostbyaddr*.

sndhostent 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.

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	This is a non-recoverable error.
NO_ADDRESS	The requested name is valid but does not have an IP address; this is not a temporary error. This means another type of request to the name server will result in an answer.

FILES

/etc/hosts

SEE ALSO

hosts(5), resolver(3), 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.

gethostent 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

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 returns 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 is not 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(4)

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

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 returns 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 is not 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(4)

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

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 returns 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 is not closed after each call to *getservbyname* or *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(4)

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

inet: 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)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

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

DESCRIPTION

The routines *inet_addr* and *inet_network* 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 addressaes 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(4), networks(4)

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

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

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

fclose(3), open(2)

DIAGNOSTICS

fopen and *freopen* return the NULL pointer 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 probably treat the *type* as if the "+" were 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

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

DIAGNOSTICS

fread and *fwrite* return 0 upon end of file or error.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

```
#include <stdio.h>

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

long ftell(stream)
FILE *stream;

rewind(stream)
```

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** 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), fread(3S), gets(3S), putc(3S), scanf(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 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

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

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

printf, fprintf, sprintf – formatted output conversion

SYNOPSIS

```
#include <stdio.h>

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

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

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

_doprnt(format, args, stream)
char *format;
va_list *args;
FILE *stream;
```

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'. All of these routines work by calling the internal routine **_doprnt**, using the variable-length argument facilities of *varargs*(3).

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

- Zero or more of following flags:
  - a '#' character specifying that the value should be converted to an "alternate form". For **c**, **d**, **s**, and **u**, conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero. For **x**(**X**) conversion, a non-zero result has the string **0x**(**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G**, conversions, the result always contains a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those conversions if a digit follows the decimal point). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.
  - a minus sign '–' which specifies *left adjustment* of the converted value in the indicated field;
  - a '+' character specifying that there should always be a sign placed before the number when using signed conversions.
  - a space specifying that a blank should be left before a positive number during a signed conversion. A '+' overrides a space if both are used.
- an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width it is 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 is done instead of blank-padding;

- an optional period '.' which serves to separate the field width from the next digit string;
- an optional digit string specifying 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 decimal, octal, or 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 MAXUINT, 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 by *putc*(3S).

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 π to 5 decimals:

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

SEE ALSO

ecvt(3), putc(3S), scanf(3S)

BUGS

Very wide fields (>128 characters) fail.

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

fclose(3S), fopen(3S), fread(3S), getc(3S), printf(3S), puts(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 output *stream*.

Neither routine copies the terminal null character.

SEE ALSO

ferror(3S), fopen(3S), gets(3S), printf(3S), putc(3S)
fread(3S) for *fwrite*

BUGS

puts appends a newline, *fputs* does not, all in the name of backward compatibility.

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 is 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.

e a floating point number is expected; the next field is converted accordingly and
f 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
```

assigns to *i* the value 25, *x* the value 5.432, and *name* contains *'thompson\0'*. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips '0123', and places the string '56\0' in *name*. The next call to *getchar* returns '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 – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. *fflush* (see *fclose*(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from *malloc*(3) upon the first *getc* or *putc*(3S) on the file. If the standard stream **stdout** refers to a terminal it is line buffered. The standard stream **stderr** is always unbuffered.

setbuf is used after a stream has been opened but before it is read or written. The character array *buf* is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, I/O is completely unbuffered. A manifest constant BUFSIZ tells how big an array is needed:

```
char buf[BUFSIZ];
```

setbuffer, an alternate form of *setbuf*, is used after a stream has been opened but before it is read or written. The character array *buf* whose size is determined by the *size* argument is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, I/O is completely unbuffered.

setlinebuf is used to change *stdout* or *stderr* from block buffered or unbuffered to line buffered. Unlike *setbuf* and *setbuffer* it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using *freopen* (see *fopen*(3S)). A file can be changed from block buffered or line buffered to unbuffered by using *freopen* followed by *setbuf* with a buffer argument of NULL.

SEE ALSO

fclose(3S), fopen(3S), fread(3S), getc(3S), malloc(3), printf(3S), putc(3S), puts(3S)

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

close(2), fread(3S), fseek(3S), f*(3S), open(2), read(2), write(2)

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
clearerr	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

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 is 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), fseek(3S), setbuf(3S)

DIAGNOSTICS

ungetc returns EOF if it can't push a character back.

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

getenv(3), ioctl(2), termcap(5), tty(4)

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*

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 superseded by *ndbm*(3), and is now implemented using ndbm.

These functions maintain key/content pairs in a data base. The functions handle very large (a billion blocks) databases and access a keyed item in one or two file system accesses. The functions are obtained with the loader option **–ldbm**.

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

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

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* returns the first key in the database. With any key *nextkey* returns the next key in the database. This code traverses 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 contains 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* returns 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

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*(4). 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 looks in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type **name** is the same as the environment string TERM, 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*(4), 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(4)

AUTHOR

William Joy