NAME

`intro` – Introduces system calls and error numbers

SYNOPSIS

```
#include <errno.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

This manual describes all UNICOS system calls. `intro`(2) contains sections on the following:

- Security privilege information
- Socket system calls
- Errors, including a listing of all error numbers

Security Privilege Information

If your UNICOS system is using the default privilege assignment lists (PALs), many of the UNICOS system calls expect that the calling process has certain privileges effective in order for the system call to execute correctly.

The man page for each affected UNICOS system call lists the privileges associated with the call. Also included is a description of what tasks or functions the associated privileges allow the system call to perform. For a list of the privileges and a general description of each privilege used on a UNICOS system, see the Security section in *General UNICOS System Administration*, Cray Research publication SG–2301.

Socket System Calls

The transport-level protocols, `tcp`(4P) and `udp`(4P), along with the network level protocol `ip`(4P), are described in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014.

A socket is a general network interface that provides programs with a uniform view of network protocol suites. The program relates to the sockets rather than the protocol; therefore, the program can use any network protocol suite (such as TCP).

Certain semantics of the basic socket abstractions are protocol-specific. Each protocol is expected to support the basic model for its particular socket type, but may, in addition, provide nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the `SOCK_STREAM` abstraction may allow more than 1 byte of out-of-band data to be transmitted per out-of-band message.

Sockets using the TCP protocol are either *active* or *passive*. Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket, the `listen`(2) system call must be used after binding the socket with the `bind`(2) system call. Only passive sockets may use the `accept`(2) call to accept incoming connections. Only active sockets may use the `connect`(2) call to initiate connections.

The Internet family, inet(4P), provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types. Transmission Control Protocol (TCP) supports the SOCK_STREAM abstraction, while the User Datagram Protocol (UDP) supports the SOCK_DGRAM abstraction. A super user may achieve a raw interface to the Internet Protocol (IP) and Internet Control Message Protocol (ICMP) by creating an Internet socket of type SOCK_RAW.

A passive socket may underspecify its location to match incoming connection requests from networks. This technique, termed *wildcard addressing*, allows one server to provide service to clients on multiple networks. To create a socket that listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified, the system assigns one. When a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the network interface through which packets are being transmitted and received. Usually, this address corresponds to the peer entity's network. UDP supports the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless, and they are normally used with the sendto (see send(2)) and recvfrom (see recv(2)) calls, although the connect(2) call may also be used to fix the destination for future packets (in which case, recv(2) and send(2) or the read(2) or write(2) system calls can be used).

A write to a raw socket must not include the IP header at the beginning of the data unless the IP_HDRINCL socket option has been set (see setsockopt(2)). A read from a raw socket always returns the IP header.

Internet Control Message Protocol (ICMP) sockets are also available through raw sockets. Refer to icmp(4P). The files are:

- TCP/IP library routines are in /lib/libc.a.
- TCP/IP include files are in the directory /usr/include.
- Symbolic names for errors are in /usr/include/errno.h.

Errors

Most system calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value, which is almost always −1; the individual descriptions specify the details. An error number is also made available in the `errno` external variable, which is not cleared on successful calls; therefore, it should be tested only after an error has been indicated. Each system call description tries to list all possible error numbers.

The following is a list of the error numbers and their names as defined in the `errno.h` header file. Tape users may receive error codes that are not documented on the system call man pages. For the descriptions of the tape daemon return values, see the *Tape Subsystem User's Guide*, Cray Research publication SG–2051.

`1 EPERM Not owner`
: Typically, this error indicates that you attempted to modify a file in a way not allowed except to its owner or a super user. It is also returned when other users try to perform actions allowed only to super users.

`2 ENOENT No such file or directory`
: Either the specified file does not exist or one of the directories in a path name does not exist.

`3 ESRCH No such process`
: No process can be found corresponding to the process that you specified.

`4 EINTR Interrupted system call`
: An asynchronous signal (such as `interrupt` or `quit`), which you have elected to catch, occurred during a system call. If execution is resumed after processing the signal, it appears that the interrupted system call returned this error condition.

`5 EIO I/O error`
: A physical I/O error has occurred. In some cases, this error may occur on a call following the one to which it actually applies.

`6 ENXIO No such device or address`
: During a read or write on a special file, a subdevice that does not exist or is beyond the limits of the device was referenced.

`7 E2BIG Arg list too long`
: Your argument list to a member of the `exec`(2) family is longer than `NCARGS` bytes.

`8 ENOEXEC Exec format error`
: A request was made to execute a file that, although it has the appropriate permissions, does not start with a valid magic number (see `a.out`(5)).

`9 EBADF Bad file number`
: Either a file descriptor refers to no open file, or a read or write request is made to a file that is open only for writing or reading, respectively.

`10 ECHILD No child processes`
: A `wait`(2) system call was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN Resource temporarily unavailable
The resource is unavailable now; later calls to the same routine may complete normally.

12 ENOMEM Not enough space
During an exec(2) or sbreak(2) system call, a program requested more space than the system could supply. This is not a temporary condition; the maximum space specification is a system parameter.

13 EACCES Permission denied
You attempted to access a file in a way not allowed by the protection system.

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

15 ENOTBLK Block device required
A call specifies something other than a block device where a block device is required (for example, in mount(2)).

16 EBUSY Device busy
You attempted to mount a device that was already mounted or to dismount a device on which there is an active file (open file, current directory, mounted-on file, or active text segment). The device or resource is currently unavailable.

This error also occurs if you try to enable accounting when it is already enabled or if you issue a restart(2) attempt when another job or process in the system is using the *jid* or any *pid* associated with the job (or process) to be restarted.

17 EEXIST File exists
A call specifies an existing file in an inappropriate context (for example, link).

18 EXDEV Cross-device link
You attempted a link to a file on another logical device.

19 ENODEV No such device
You attempted to apply an inappropriate system call to a device (for example, to read a write-only device).

20 ENOTDIR Not a directory
A call specifies a nondirectory where a directory is required (for example, in a path prefix or as an argument to chdir(2)).

21 EISDIR Is a directory
You attempted to write on a directory.

22 EINVAL Invalid argument
The call contains an argument that is not valid such as the dismounting of a nonmounted device, the mention of an undefined signal in signal(2) or kill(2), or the reading or writing of a file for which lseek(2) has generated a negative pointer. This error is also set by the math functions described in the (3) entries.

23 ENFILE File table overflow
The system file table is full and temporarily cannot accept more open(2) calls.

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

25 ENOTTY Not a typewriter
You attempted to use an ioctl(2) request with a file that is not a character special file.

26 ETXTBSY Text file busy
Either you attempted to execute a pure-procedure program that is currently open for writing or reading, or you attempted to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
The size of a file exceeds the maximum file size or the process size set by the ulimit(2) system call.

28 ENOSPC No space left on device
During a write(2) to an ordinary file, the free space left on the device was exhausted.

29 ESPIPE Illegal seek
You issued an lseek(2) to a pipe. This is not allowed.

30 EROFS Read-only file system
You attempted to modify a file or directory on a device mounted as read-only.

31 EMLINK Too many links
You attempted to make more than LINK_MAX links to a file.

32 EPIPE Broken pipe
A write was performed on a pipe for which no process exists to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

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

34 ERANGE Result too large
The value of a function in the math package (3) cannot be represented within machine precision.

35 ENOMSG No message of desired type
This message is used by internal tools. If you receive this message, contact your system administrator.

36 EIDRM Identifier removed
This message is reserved for future use. If you receive this message, contact your system administrator.

37 ECHRNG Channel number out of range
This message is reserved for future use. If you receive this message, contact your system administrator.

38 EL2NSYNC Level 2 not synchronized
This message is reserved for future use. If you receive this message, contact your system administrator.

39 EL3HLT Level 3 halted
: This message is reserved for future use. If you receive this message, contact your system administrator.

40 EL3RST Level 3 reset
: This message is reserved for future use. If you receive this message, contact your system administrator.

41 ELNRNG Link number out of range
: This message is reserved for future use. If you receive this message, contact your system administrator.

42 EUNATCH Protocol driver not attached
: This message is reserved for future use. If you receive this message, contact your system administrator.

43 ENOCSI No CSI structure available
: This message is reserved for future use. If you receive this message, contact your system administrator.

44 EL2HLT Level 2 halted
: This message is reserved for future use. If you receive this message, contact your system administrator.

45 EDEADLK Deadlock situation detected/avoided
: A deadlock situation was detected and avoided. This error pertains to file and record locking.

46 ENOLCK No record locks available
: The setting or removing of record locks on a file cannot be accomplished, because no more record lock entries are left in the system.

47 EINVFS Not allowed on this file system
: An operation was performed on a file system type that does not support that operation.

50 EFILECH File changed
: Either a file referenced by the restart file has been changed since the restart file was created, or a file residing remotely on a network file system (NFS) has changed.

51 EFILERM File removed
: Either a file needed for the checkpointing or restarting of a job or process has no links to it, or a file residing remotely on a network file system (NFS) has been removed.

52 ERFLOCK Recovery of file lock would block
: In `restart`(2), record locks owned by the processes to be restarted could not be recovered, because record locks owned by currently existing processes have one or more of the target file regions already locked.

53 ENOSDS Unable to recover SDS space
: (Cray PVP systems) During an attempt to recover a job by using `restart`(2), the secondary data segment (SDS) requirement of this job exceeded the current availability of SDS.

54 EFILESH Fair-share scheduler controls file.
: An unlinked regular file needed for the checkpointing of a job or process is in use by one or more processes outside the set of processes to be checkpointed.

55 EMALFORMED Malformed process collection
: The target set specified by a chkpnt(2) request does not represent a completely contained set. For example, if a process is using a file that is currently opened by more than one process, and if all of the processes that have the file opened are not within the specified process set, this error occurs.

56 EFOREIGNFS Foreign file system
: An operation that is supported only on local file systems was attempted on a nonlocal (foreign) file system.

60 EQUSR User file/inode quota limit reached
: A program writing a file under your current user ID has reached a file or inode quota limit. For a temporary solution, remove some of your files so that additional space is available or move the files to a file system that has sufficient quota authorization. For a more permanent solution, contact your system administrator and request additional space.

61 EQGRP Group file/inode quota limit reached
: A program writing a file under your current group ID has reached a file or inode quota limit. For a temporary solution, remove some of your files so that additional space is available or move the files to a file system that has sufficient quota authorization. For a more permanent solution, contact your system administrator and request additional space.

62 EQACT Account file/inode quota limit reached
: A program writing a file under your current account ID has reached a file or inode quota limit. For a temporary solution, remove some of your files so that additional space is available or move the files to a file system that has sufficient quota authorization. For a more permanent solution, contact your system administrator and request additional space.

66 EREMOTE Object is remote
: No explanation is available for this message.

74 EMULTIHOP Multihop attempted
: No explanation is available for this message.

75 ESHMA Process has shared memory segment attached
: Process with attached shared memory segments (CRAY T90 series systems only) cannot be checkpointed.

90 EPROCLIM Process limit exceeded
: This message is returned when a user exceeds the fair-share scheduler process limit.

91 EMEMLIM Memory limit exceeded
: This message is returned when a user exceeds the fair-share scheduler memory limit.

92 EDISKLIM Disk limit exceeded
: Your job-based or process disk limit has been exceeded.

93 ETOOMANYU Too many users
: You exceeded the system compile-time definition of NUSERS, which defaults to 200.

94 ENAMETOOLONG Filename too long
: Either the size of a path name string exceeds the maximum allowed, or a path name component exceeds the maximum and automatic component truncation is not supported for the file system of the file that the component names.

95 ENOSYS Function not implemented
: An attempt was made to use a function that is not available in this implementation.

96 ENOTEMPTY Directory not empty
: This error code is defined for compatibility with the POSIX 1003.1 standard, but it is not used.

97 ERENAMESELF Attempt to rename a link to itself
: This error code is used internally in the rename(2) system call. It is never returned to a user program.

98 ELOOP Too many symbolic links
: Too many symbolic links were encountered when a path name was translated.

UNICOS issues the following TCP/IP network, socket interface error codes:

128 EWOULDBLOCK Operation would block
: An operation that would cause a process to block was attempted on an object in nonblocking mode (see ioctl(2)).

129 EINPROGRESS Operation now in progress
: An operation that takes a long time to complete (such as connect(2)) was attempted on a nonblocking object (see ioctl(2)).

  For example, if you issued a connect(2) system call on a nonblocking socket, you would normally receive this error. Although the connection is not yet established, the application program does not need to do anything special as the connection is being established asynchronously. (The connection is usually established before you can issue another system call.) If, however, you issued a write(2) (or similar) system call before the connection is actually established, the EWOULDBLOCK error would be returned.

130 EALREADY Operation already in progress
: An operation was attempted on a nonblocking object that already had an operation in progress. The application has to wait until the nonblocking operation completes.

131 ENOTSOCK Socket operation on non-socket
: Certain system calls (for example, getpeername(2)) operate only on sockets. Such an operation was attempted on a nonsocket descriptor.

132 EDESTADDRREQ Destination address required
: A required address was omitted from an operation on a socket. Supply the appropriate address.

133 EMSGSIZE Message too long
: A message sent on a socket was larger than the internal message buffer. Shorten the message.

134 EPROTOTYPE Protocol wrong type for socket
The protocol specified does not support the semantics of the socket type requested. For example, you cannot use the DARPA Internet UDP protocol with type SOCK_STREAM. Check to be sure that the operation attempted and the type of socket match.

135 ENOPROTOOPT Bad protocol option
A bad option was specified in a getsockopt or setsockopt system call (see getsockopt(2)). Check the various arguments and correct as necessary.

136 EPROTONOSUPPORT Protocol not supported
Either the specified protocol has not been configured into the system, or no implementation for it exists. Check the protocol argument on the system call.

137 ESOCKTNOSUPPORT Socket type not supported
Either support for the specified socket type has not been configured into the system, or no implementation for it exists. Check the protocol argument on the system call.

138 EOPNOTSUPP Operation not supported on socket
For example, trying to use accept(2) to accept a connection on a datagram socket (type SOCK_DGRAM) is not supported. Check the parameter on the socket(2) system call that created the socket.

139 EPFNOSUPPORT Protocol family not supported
Either the specified protocol family has not been configured into the system, or no implementation for it exists. Check the protocol argument on the system call.

140 EAFNOSUPPORT Address family not supported by protocol family
An address incompatible with the requested protocol was used (for example, you cannot always use PUP Internet addresses with DARPA Internet protocols). For TCP/IP protocols, the address family is AF_INET.

141 EADDRINUSE Address already in use
Normally, only one socket is allowed to be bound to a local address (Internet address and port number). You sometimes receive this message when you are using the bind(2) system call to bind a socket to a local address. Use the netstat(1B) command to find out whether the address is already bound.

142 ENETUNREACH Network is unreachable
A socket operation was attempted on an unreachable network. Check the destination address. If it is valid, the network is currently unavailable.

143 ENETRESET Network dropped connection on reset
The connected host crashed and rebooted. Restart your program.

144 ECONNABORTED Software caused connection abort
A connection abort was internal to the local host. Retry the command once. If that fails and the local host is a Cray Research mainframe, see your system support staff.

145 ECONNRESET Connection reset by peer
: A connection was forcibly closed by a peer. This action normally results when the peer executes a shutdown(2) system call.

146 ENOBUFS No buffer space available
: An operation on a socket or pipe was not performed because the system lacked sufficient buffer space. Retry the operation. Also use the netstat(1B) command to see whether you are running out of memory. If you frequently run low on memory, ask your site analyst to configure the system with more mbufs.

147 EISCONN Socket is already connected
: Either a connect(2) request was made on an already connected socket, or a sendto or sendmsg (see send(2)) request on a connected socket specified a destination other than the connected party.

148 ENOTCONN Socket is not connected
: A request to send or receive data was disallowed because the socket is not connected. Issue a connect(2) system call.

149 ESHUTDOWN Cannot send after socket shutdown
: A request to send data was disallowed because the socket had already been shut down with a previous shutdown(2) system call. All send(2), sendto(2), and write(2) system calls fail with this error after a shutdown(2) system call has been issued.

150 ETOOMANYREFS No free TP references available
: A connect(2) request or listen(2) request failed because no free TP (transport) reference blocks are available. A timer is set whenever a reference block is freed; the reference may not be reused until the timer expires to ensure connection uniqueness within the system. (The time-out period depends on the communication protocol.)

  This error may be caused by the maximum number of active connections allowed by the system or by many connections being established and closed in quick succession. In either event, try later at an appropriate time.

151 ETIMEDOUT Connection timed out
: A connect(2) request failed because the connected party did not respond properly within a specified period of time. (The time-out period depends on the communication protocol.) Make sure the remote server process is running. If it is, try later at an appropriate time.

152 ECONNREFUSED Connection refused
: The connection could not be made, because the target machine actively refused it. You are probably trying to connect to a service that is inactive on the remote host. Make sure that the service is actually running before retrying.

153 EHOSTDOWN Host is down
: A socket operation failed because the destination host was down. First, recheck the address to be sure it is correct. If it is, retry the operation later when the host is available.

154 EHOSTUNREACH Host is unreachable
A socket operation was attempted on an unreachable host. First, recheck the address to be sure it is correct. If it is, retry the operation later when the host is available.

155 EADDRNOTAVAIL Cannot assign requested address
This message normally results from an attempt to create a socket with an address not on the local machine. Check the parameters, especially the part number, on the socket(2) system call.

156 ENETDOWN Network is down
A socket operation encountered a dead network. Retry the operation when the network is available.

UNICOS issues the following BMX-TSS driver interface error codes:

200 ETPDCNF Tape open rejected due to device configuration
The device is not configured. You tried to open a tape device, and the system is not configured for tapes, or that particular device is not configured up. Contact your system administrator.

201 ETPDOPN Tape open rejected, already open to another
The device is already in use by another process. You tried to issue an open(2) or other request to a tape device which is already assigned to another user. Use a different tape device, or contact your system administrator.

202 ETPDABN Tape I/O request with abnormal status set
An I/O request completed abnormally due to a previous or current error. Check your tape.msg file for more information on the error.

203 ETPDNRW No write ring on tape device
The device requires a write ring. This message is obsolete at UNICOS 7.0.

204 ETPDBDF Bad data returned on tape read
The read(2) function returned incorrect data from the tape. This probably means you have a bad tape.

205 ETPDEOV Tape end of volume
This message is obsolete at UNICOS 7.0.

206 ETPDCLEAR Tape cleared by operator
The device has been cleared by the operator with a tpclr(8) command. Contact your system administrator.

207 ETPDEOF End-of-file tape mark was read.
A user tape mark has been read indicating that end-of-file was reached. This is an informative message, and no action is required.

208 ETPDNODEM Tape daemon is not active.
The tape daemon has gone down or not been started yet. Contact your system administrator.

209 ETPDBUFZ User buffer size is not valid.
You have specified a buffer size that is not valid for either tape list I/O or unbuffered data (using the -U option on the tpmnt(1) command). The buffer size must be a multiple of 4096 bytes.

- On input requests, the size must exceed the sum of each list entry rounded up to a multiple of 4096 bytes.
- On output requests, the size must exceed the sum of each list entry rounded up to a multiple of 4096 bytes.

Check the buffer size that you are using against what you specified on `tpmnt`.

`210 ETPDRWE A read-after-write or write-after-read occurred.`
Either a read has been requested after a write, or a write has been requested after a read. Both sequences are illegal with tape. Check your program.

`211 ETPDLIST Error in list`
The tape list structure contains one of the following errors:

- You did not specify any entries for tape list I/O.
- You specified an invalid state.
- The byte count is either 0, or it exceeds the maximum block length.

Correct the tape list structure, and then continue processing.

`212 ETPDUERR User error, only close allowed`
Your job can only issue a `close`(2) request due to a previous error. Check your program.

`213 ETPDMBS Maximum tape block size exceeded`
This message is obsolete at UNICOS 7.0.

`214 ETPDLBK Large block tape error`
The block requested is too large for a model E machine; either it is larger than the system maximum, or it is larger than the maximum specified on the `-b` option of your `tpmnt`(1) command. Specify a smaller size block.

`215 ETPDACKERR Acknowledge error before continuing.`
You received a previous error while using asynchronous I/O. If you are executing a Fortran program, the Fortran libraries handle this function. Otherwise, you must acknowledge the error by issuing an `ioctl`(2) system call for `TPC_ACKERR` before any other requests will be honored.

`216 ETPDNOSYSBF No system buffers are available.`
The tape subsystem was unable to obtain the memory needed for the tape subsystem I/O buffers. Contact your system support staff.

`217 ETPDSTOP The IOP is stopped.`
An I/O request was terminated because the tape subsystem is being restarted. Reissue the request later.

`218 ETPDMAXDEVUP Maximum tapes configured up will be exceeded`
A request to configure a device up was terminated because the current number of devices configured up is at the system limit. This limit is defined by the system parameter, `TAPE_MAX_CONF_UP`. Contact your system support staff.

219 ETPD_PK_BADLEN Packet length is not valid.
A request to send a packet to an IOP or channel contains a packet length that is not within the valid range, 3 – EPAK_MAXLEN. Correct the packet length and reissue the request.

220 ETPD_PK_NOT_ALLOWED The packet request is not valid.
An ioctl(2) request to send a packet to an IOP or channel device is not valid for that device type. Correct the IOP request packet or reissue the request to the correct device.

221 ETPD_PK_SEND Error sending packet
A request could not be sent to an IOP. Contact your system support staff.

222 ETPD_PK_TIMEDOUT The IOP request timed out.
The time-out period expired without receiving a response from the IOP. Contact your system support staff.

223 ETPD_PK_CHAN_UP Channel is configured up
An attempt to open a channel device failed because the channel is configured up. Diagnostic requests to a channel device can be issued only to a channel that has been configured down by the tape subsystem.

Configure the channel down with the tpconfig(8) command and reissue the open request.

224 ETPD_PK_CHAN_OPENED The channel is already open.
An attempt to modify the configuration of a channel failed because the channel device is open for diagnostic use. Wait until the channel device is closed and try again.

225 ETINVCTL The ioctl request is not valid.
The ioctl(2) request issued is not valid. Correct the request and reissue it.

226 ETPD_BAD_REQT The request issued to the IOP or device is not valid.
An IOP or device request was terminated either because the contents or the format of the request is incorrect or because the sequence of requests is not valid. Contact your system support staff.

227 ETPD_BLANK_TAPE A blank tape was detected.
If this request was issued to an ER90 device, the tape operation was terminated because the command cannot be issued to a device with a blank tape loaded.

If this request was issued to a block multiplexer device, the command was terminated because it cannot execute when the tape is positioned before a blank portion of the tape.

228 ETPD_NOT_OPER The device is not operational.
A device request failed because of a hardware error. Additional information can be obtained from the error log. Contact your system support staff.

229 ETPD_NOT_READY The device is not ready.
A device request failed because the device is not ready. Switch the device to the ready state and retry the request.

230 ETPD_EOT End of tape detected
A device request could not complete because the end of tape was detected. This message is used internally and is not returned to the user.

231 ETPD_DATA_ERROR An unrecoverable data error occurred.
A read or write request failed because of a permanent read or permanent write error. Contact your system support staff.

232 ETPD_MEDIA Media is not supported
The configuration of the cassette is not supported.

233 ETPD_EOR End of recording was detected
A request failed because the EOR was detected. The EOR is a recorded entity that indicates the end of recording for a partition. Either reposition the tape before the data or record some data at the current position and then reissue the request.

234 ETPD_LGPS Logical position has not been established
Your request failed because the logical position had not been established.

If file positioning was turned off using the `-z` option on the `tpmnt`(1) command, you received this message because a request to position to an absolute track address immediately following the tape open was omitted. Position the tape with a `TPC_DMN_REQ ioctl` request with a subrequest type of `TR_PABS` and then reissue the request.

If you did not use the `-z` option on the `tpmnt` command, you received this message because a problem has occurred within the tape subsystem. Contact your system support staff.

235 ETPD_EOM End of media was detected
The end of the media was detected. Correct the problem and retry. If you receive this message again, contact your system support staff.

236 ETPD_SYSTEM A tape driver error occurred.
A tape driver software error occurred. Contact your system support staff.

237 ETPD_DEVICE A device error occurred.
The tape driver received a response that was not valid from the tape device. Contact your system support staff.

238 ETPD_FORMAT This volume format is not supported.
Either an ER90 device was unable to format even one partition on a volume, or the format of the volume is not supported. Contact your system support staff.

239 ETPD_FILETYPE The tape file type is not valid.
This error is returned to the tape daemon if the current file section is a byte stream file section, but the tape daemon issued a blocked I/O request. This error is not returned to the user.

240 ETPD_NO_CASSETTE A cassette is not loaded.
You issued a request that requires a cassette to be loaded to a drive in which a cassette is not currently loaded. Issue a `tpmnt` (1) command and then reissue the original request.

241 ETPD_TAPE_ADDR The specified tape address is not valid.
A request to position to an absolute track address failed because the address specified does not exist on the currently mounted cassette. Correct the request and reissue it.

242 ETPD_DEVDOWN The device must be configured up.
A request was issued to a downed device, but the request requires that the device be configured up. Use the tpconfig(8) command to configure the device up and then reissue the request.

243 ETPD_NOT_BOF Must be at the beginning of file
A device request was terminated because the tape is not positioned at the beginning of a file section.

Position the tape with TPC_DMN_REQ positioning and then reissue the device request.

244 ETPD_TAPE_ERROR A tape error occurred.
A request failed because of a media problem. If an ER90 device was used, the ER90 was unable to locate a byte or block due to a tape fault or to an incorrect tape format. Contact your system support staff.

245 ETPD_DEV_HUNG Device is hung
A response was not received from a tape device. Contact your system support staff.

246 ETPD_MAX_IOREQT Exceeded I/O request size maximum
You issued an I/O request that exceeds the tape subsystem, IOP, or device limits. Correct the request and then reissue it.

247 ETPD_ODD_BYTES Cannot issue an odd byte I/O request
An I/O request was terminated because the previous I/O request output or input an odd number of bytes. An odd byte I/O request is only valid at the end of a file.

Check the requests and correct, as needed. Then reissue the corrected requests.

248 ETPD_BLKSIZ_DIFF Blocks must be the same size within a file.
An I/O request was terminated because the previous I/O request output or input a block of a size shorter than the block size defined for the file section. All blocks within a file section must be the same size, excluding the last block in the file section.

Check the requests and correct, as needed. Then reissue the corrected requests.

249 ETPD_POSACC_ERR Position cannot be accessed
A request was terminated because it attempted to access data at an odd-byte memory address.

Position the tape with TPC_DMN_REQ positioning and then reissue the device request.

UNICOS issues the following communications driver error codes:

250 ELATE I/O request timeout
A read(2), reada(2), write(2), writea(2), or listio(2) request to execute I/O on a communications channel (NSC HYPERchannel, FEI-3, other low-speed device, or HSX) has resulted in no I/O for a certain interval of time. This interval is determined by the type of channel and IOS model.

251 ENSC NSC HYPERchannel error on write
No explanation is available for this message.

UNICOS issues the following security violation error codes:

`300 ESYSLV Security level violation`
: The specified security level falls outside the allowed security level range of the process, file system, or UNICOS system. See your security administrator.

`301 EREADV Security read violation`
: An attempt to gain read access to a file has failed because your active security level is less than the file's security level. Raise your active security level to be greater than or equal to the file's security level. If you are not authorized to raise your security level to an appropriate value, see your security administrator.

`302 EWRITV Security write violation`
: An attempt to gain write access to a file has failed because your active security label is not equal to the file's security label. Change your active security label to match the file's security label. If you are not authorized to change your security label to the appropriate value, see your security administrator.

`303 EEXECV Execute security violation`
: An attempt to gain execute/search access to a file has failed because your active security level is less than the file's security level. Raise your active security level to be greater than or equal to the file's security level. If you are not authorized to raise your security level to an appropriate value, see your security administrator.

`304 ECOMPV Security compartment violation`
: An attempt to gain access to a file has failed because your active security compartments do not include all of the file's compartments. For write access, change your active compartments to equal the file's compartments. For read or execute/search access, change your active compartments to include the file's compartments. If you are not authorized to change your active compartments to an appropriate value, see your security administrator.

`305 EMANDV Security mandatory access violation`
: Your active security label does not permit access to a file. For write access, change your active security label to match the file's security label. For read or execute/search access, change your active security label to include the file's security label. If you are not authorized to change your security label to an appropriate value, see your security administrator.

`306 EOWNV Security owner violation`
: You are not authorized to access this file. You must be the file's owner or an appropriate administrator to perform the requested file operation. See your security administrator.

`307 ELEVELV Security level range violation`
: The specified security level falls outside the allowed security level range of the process or file system. See your security administrator.

`308 ESECADM Unauthorized user`
: You are not authorized to make this request.

309 EFLNEQ Security mount violation
An attempt to allocate space on a file system has failed because your active security level falls outside the allowed security level range of the file system. Change your active security level to be within the bounds of the file system. If you are not authorized to change your active security level to an appropriate value, see your security administrator.

310 ENOTEQ Security buffer violation
This error code is unused.

311 EPERMIT Security permission violation
You do not possess the appropriate authorization(s) to perform the requested function. See your security administrator.

312 EACLV Access list violation
This error code is unused.

313 ENOACL No acl list
This error code is unused.

314 ESLBUSY Security log in use
A request to open security log device /dev/slog for reading was refused because the device has already been opened for reading by another process. This prohibits two versions of the security log daemon (slogdemon) from operating simultaneously. The slogdemon should be the only process allowed to request an open on /dev/slog. See your security administrator.

315 ESLNXIO Security log mode violation
A request to open security log device /dev/slog was refused because it attempted the open with write permission. The security log daemon (slogdemon) should be the only process to request an open on /dev/slog, and then only for reading. See your security administrator.

316 ESLFAULT Security log read violation
During the transfer of data from the security log device /dev/slog to the disk-resident security log file, a call to a copy out routine returned an error indicating a problem in the memory addressing of the read buffer. See your security administrator.

317 ESLNOLOG Security log configured SLGOFF
This error code is unused.

318 EINTCLSV Security class violation
The requested integrity class falls outside the allowed integrity class range of the process, or the UNICOS system. See your security administrator. The use of integrity class values is no longer supported.

319 EINTCATV Security category violation
The requested category is not included in the allowed categories of the process, or the UNICOS system. See your security administrator.

320 ENONAL No network authorization list (NAL)
This error code in unused.

321 EMNTCMP Security mount compartment violation
This error code is unused.

322 EFIFOV Security FIFO violation
This error code is unused.

323 EAPPNDV Security append violation
This error code is unused.

324 ETFMCATV Security multicategory violation
This error code is unused.

325 ECOVERT Covert channel condition
A file has been created in a wildcard directory by a user who does not own that directory. This is an informative message; you are not required to take corrective action.

326 ERCLSFY Security label reclassify violation
This error code is unused.

327 EPRLABEL Security label printing disabled
This error code is unused.

328 ENONSECURE Security is not enabled
This error code is unused.

329 ESECFLGV Security flag violation
A request to set file security flags has failed because the requested flags are not allowed. Specify only security flags that are available on the UNICOS system. See your security administrator.

330 EHOSTNAL Host not authorized in NAL
Access to or from an unauthorized host or workstation was attempted. The host is not authorized in the network authorization list (NAL). See your security administrator.

331 ESLVLNAL Security level outside host's range
A security level was detected outside of the security level range authorized for the host in the network authorization list (NAL). The kernel detected this condition when processing the Internet Protocol (IP) security option associated with a datagram. See your security administrator.

332 ESCMPNAL Security compartment outside host's range
A security compartment was detected outside of the security compartment range authorized for the host in the network authorization list (NAL). The kernel detected this condition when processing the Internet Protocol (IP) security option associated with a datagram. See your security administrator.

333 EMODENAL Illegal transmission for host (NAL)
An illegal mode (send or receive) of transfer was attempted by a host or workstation. See your security administrator.

334 ESLVNIF Security level outside network I/F
A security level was detected outside of the security level range authorized for the UNICOS network interface (I/F). The kernel detected this condition when processing the Internet Protocol (IP) security option associated with a datagram. See your security administrator.

335 ESCMPNIF Compartment level outside network I/F
A security compartment was detected outside of the security compartment range authorized for the host in the UNICOS network interface (I/F). The kernel detected this condition when processing the Internet Protocol (IP) security option associated with a datagram. See your security administrator.

336 ESOCKLVL Security level change of SLS tried
An illegal attempt was made to change the security level of a single-level socket (SLS) connection. Except for a privilege granted by your security administrator (for example, the network file system (NFS)), all socket connections are created as SLS. See your security administrator.

337 ESOCKCMP Compartment change of SLS attempted
An illegal attempt was made to change the security compartment of a single-level socket (SLS) connection. Except for a privilege granted by your security administrator (for example, the network file system (NFS)), all socket connections are created as SLS. See your security administrator.

338 ENFSAUTH Invalid NFS authentication credential
The proper authentication credentials were not passed to the network file system (NFS). Check your authentication credentials, or see your security administrator.

339 ESLVLNRT Security level violation network route
A security level was detected outside of the security level range authorized for the network route selected, or a route with the correct sensitivity label could not be found. The kernel detected this condition when selecting routes. See your security administrator.

340 ESCMPNRT Compartments violation for network route
A security compartment was detected outside of the security compartment authorized for the network route selected, or a route with the correct sensitivity label could not be found. The kernel detected this condition when selecting routes. See your security administrator.

341 EBADIPSO Bad IP security option
An illegal IP security option was detected by the kernel. The kernel performs integrity and security checks against each IP security option received. See your security administrator.

342 ENOIPSO IP security option missing
An IP security option did not accompany an incoming datagram. The kernel detects this condition by using IP security option information defined in the network authorization list (NAL) for each host/workstation. See your security administrator.

343 ESLVLMAP Security level mapping error
A translation error was detected when mapping the security level (between UNICOS form and network form). When necessary, the kernel translates (using the network authorization list (NAL)) the UNICOS security label to the network security label (for outgoing datagrams), and translates the network security label to the UNICOS security label (for incoming datagrams). See your security administrator.

344 ESCMPMAP Compartment mapping error
: A translation error was detected when mapping the security compartment (between UNICOS form and network form). When necessary, the kernel translates (using the network authorization list (NAL)) the UNICOS security label to the network security label (for outgoing datagrams), and translates the network security label to the UNICOS security label (for incoming datagrams). See your security administrator.

345 EAUTHFLG Authority flag violation
: An authority protection violation was detected for either an incoming or outgoing datagram with a Basic Security option. See RFC 1108.

346 EIPSOMAP No map for security option
: No translation table was available to translate the security label for a given host connection. The kernel could access the mapping table identified in the network authorization list (NAL) for the host. See your security administrator.

UNICOS issues the following data migration facility error codes:

350 EDMRNOLF Not offline file; dmofrq system call
: The file is not an offline file.

An invalid parameter was specified in the dmofrq(2) system call.

352 EDMRWFT Incorrect file type; dmofrq system call
: An incorrect file type was specified with the m or M option of the dmofrq(2) system call.

353 EDMRNSD Device does not match; dmofrq system call
: The device is incorrect.

354 EMASS Error occurred in FILESERV storage management system.
: A FILESERV error occurred. Contact your system support staff.

355 EDMOFF Data management system is off
: The data management system is not configured.

357 EOFFLIN File offline, no automatic retrieval
: The file is offline and automatic retrieval is not selected. Select automatic retrieval by setting dmmode to 1. (See sh(1) and csh(1).)

358 EOFLNDD File offline, daemon not available
: The file is offline and the daemon is not available.

360 EOFLNNR File offline, currently not retrievable
: The file is offline and temporarily cannot be retrieved. You may need to see your system administrator for help.

361 EOFLRIN File offline, retrieval interrupted
: Retrieval was in process and the user interrupted the process. (The retrieval might continue.)

362 EOFSPACE File offline, not enough space to retrieve it
: The file is offline and there is not enough space in the file system to retrieve it.

363 EOFQUOTA File offline, retrieval would exceed disk space quota
The file is offline, and retrieval is not allowed, because this would exceed the user file quota, the group file quota, or the account file quota.

365 EDMTRSTD Function reserved for trusted subject only
The caller issued a dmofrq(2) migrate/unmigrate subfunction request, but is not a trusted subject. The data migration daemon is the sole user of this type of request; no user process should ever receive this error.

367 EDMOFRQ Invalid or inconsistent dmofrq system call parameters
The caller issued a dmofrq(2) system call, but one or more of the parameters are illegal or inconsistent for the requested subfunction. This call is used solely by components of the data migration facility; no user process should ever receive this error.

368 EDMNBLK Nonblocking recall; data recall from offline media is pending
An open system call (with the O_NONBLOCK flag set) failed on a migrated file in which the data is being recalled from an offline media. This indicates that the file is being recalled asynchronously.

371 ENOIDMAP Named ID map not found
A reference to an ID map in the kernel failed because the specified map was not found.

372 EMAPINUSE Named ID map is in use (reference count nonzero)
An attempt to remove an ID map from the kernel failed because an ID mapping domain is still referencing it.

373 EMAPTYPE Unknown ID map type
An attempt to add or delete a map in the kernel failed because the ID map type is not UID or GID.

374 EDUPMAPNAME Duplicate ID map name
An attempt to add an ID map to the kernel failed because there is already an ID map with the same name in the kernel.

375 EGIDMAPSIZE Bad group ID map size
An attempt to add a GID map to the kernel failed because the size of the GID map is not a multiple of the size of a GID map entry.

376 EBADDOMAIN ID mapping domain address does not make sense
An attempt to add or delete an ID mapping domain in the kernel failed because the Internet addresses specified for the mapping domain do not make sense. Either the upper address of the range is numerically less than the lower address, or the address mask is NULL.

377 ENODOMAIN ID mapping domain not found
An attempt to delete an ID mapping domain from the kernel failed because it was not found.

378 EBADADDR ID mapping domain overlap error
An attempt to add an ID mapping domain to the kernel failed because the specified domain overlaps with an existing domain.

379 ENOUMAPENTRY User ID map entry not found
An attempt to delete a user's entry from an ID map failed because the entry was not found.

380 EREMOTEUID Remote hash not found for user ID map entry, ID mapping disabled
An attempt to delete a user's entry from an ID map failed because after the local UID hash pointer was found and removed, the remote UID hash pointer was not found. The hash table for this map is corrupted.

381 EUIDTYPE Unknown user ID map entry type
An attempt to add or delete an entry in a user ID map failed because the type of the entry (which indicates the number of groups in the groups list for this entry) was not valid.

382 EDUPUMAPENTRY Duplicate user ID map entry
An attempt to add an entry to a user ID map failed because the entry is already in the ID map.

383 ESAMEIDMAP This ID mapping domain is unnecessary
No explanation is available for this message.

384 EMAPTHRUIDMAP A special ID map is already defined
No explanation is available for this message.

385 EMAPTHRUUID Special ID map entry (luid != ruid)
No explanation is available for this message.

386 ENOKRBADDR No Kerberos validated address found
No explanation is available for this message.

387 EDUPKRBADDR Duplicated Kerberos validated address found
No explanation is available for this message.

388 EKRBADDRINUSE Kerberos address reference count nonzero
No explanation is available for this message.

393 EIDMADDR Address not found in ID mapping domains
An NFS request failed because the address of the server is not in the ID mapping domains and ID mapping is enabled.

394 ESTALE Stale NFS file handle
An NFS request failed because the information the client has for the remote file system is no longer valid. The NFS file system must be remounted.

395 ERPCCDRES Cannot decode RPC results
An NFS request failed because the RPC results cannot be decoded.

396 ERPCCDARGS Cannot decode RPC arguments
An NFS request failed because the RPC arguments cannot be decoded.

397 ERPCCANTSEND Unable to send RPC request
The RPC request could not be sent.

398 ERPCAUTH RPC authentication error
An NFS request failed because an NFS authentication error occurred.

400 EPKI_NO_PACKETS No packets are available.
The packet driver does not have any response packets on its packet queue. Retry later.

401 EPKI_PACKET_LOST A packet was discarded.
The packet driver discarded a packet received from an IOP. Either the packet interface had not been enabled or there was not an available packet entry on the packet queue when a packet was received.

402 EPKI_TRUNCATED A packet was truncated.
This message is obsolete at UNICOS 8.0.

403 EPKI_TOO_LARGE The packet size exceeds the user buffer size.
The next packet on the packet driver queue is larger than the amount of memory allocated for the packet in the receive request (pki_nbytes).

Allocate a larger block of memory for the packet and reissue the request.

404 EPKI_ASYNCH_LIM Asynchronous response limit exceeded
You have issued a request to enable asynchronous responses that attempts to enable more asynchronous responses than allowed by the system limit.

This limit is defined, when the packet driver is started, in the packet driver configuration file, with parameter MAX_ASYNC. If this limit is not specified in the configuration file, the asynchronous response limit defaults to 5.

The value may be changed after the packet driver has been started with the ipi3_option(8) or hpi3_option(8) command.

405 EPKI_INVAL_CODE The request code is not valid.
The request packet specified was not valid. Contact your system support staff.

406 EPKI_NOT_ENABLED The packet interface has not been enabled.
The packet interface must be enabled before attempting to register a signal (PKI_SIGNO), send a packet (PKI_SEND), or receive a packet (PKI_RECEIVE).

The packet interface is enabled with the following:

```
ioctl, PKI_ENABLE
```

407 EPKI_REQ_LIM Maximum IOP request limit exceeded
The number of packets sent by the user, but not yet received, is at the packet driver limit.

- For IOP devices, the request limit is always 1.
- For IPI-3 devices, this limit is defined when the packet driver is started, in the packet driver configuration file, with parameters MAX_STK_COUNT and MAX_NON_CMDLST. If the limit is not specified in the configuration file, the request limit defaults to 10. This value may be changed after the packet driver has been started with command ipi3_option(8) or hpi3_option(8).

408 EPKI_BAD_RESYNC The resynchronization code is not valid.
The packet specified in the PKI_SEND request contains a resynchronization code that does not match the resynchronization code of the last command list response received from the IOP.

Correct the PKI_SEND request and reissue it.

409 EPKI_NO_START The IOP driver has not been started.
A packet cannot be sent to an IOP because the IOP driver has not been started.

Start the IOP driver with the ipi3_start(8) or hpi3_start(8) command and then reissue the send request.

410 EPKI_CF_TYPE A configuration type specified is not valid.
A configuration type statement specified in the packet driver configuration file is not valid.

Valid configuration type statements begin with the - character and are followed by one of the following strings: IOPS, CHANNELS, SLAVES, DEVICES, or OPTIONS.

411 EPKI_PARM_ERR The configuration definition specified is not valid.
An error was found in the packet driver configuration file.

412 EPKI_DEV_LIM Exceeded device limits
The requests exceeded the maximum number of IOP devices allowed. The maximum number is limited to MAX_IOPS. Contact your system support staff.

413 EPKI_IOS_ERR An IOS error occurred on an IPI-3 request.
An IOP request did not complete successfully. Contact your system support staff.

414 EPKI_REQT_TYPE The request is not valid for the device type.
The packet specified in a send request (PKI_SEND) is not valid for the device it was issued to. Correct the PKI_SEND request and then reissue it.

415 EPKI_IOCTL_REQT The ioctl request is not valid for the device type.
The ioctl(2) request is not valid for the device it was issued to. Correct the request and then reissue it.

416 EPKI_IOP_SEND Unable to send the request to the IOP
The packet driver was unable to send a packet to an IOP. Contact your system support staff.

417 EPKI_ACTIVE_IOP An IOP is active.
The request could not be processed because an IOP device is open. Wait for all IOP devices to be closed and then reissue the request.

418 EPKI_DEVS_ACTIVE Device(s) on IOP are active
The IOP driver could not be stopped because a device is open.

Wait for the device to be closed and then reissue the request.

419 EPKI_NOT_CONF The driver has not been configured into the system.
IPI-3 packet driver support has not been built into the current system. Contact your system support staff.

420 EPKI_SYS_ERROR Packet driver error
An IPI-3 packet driver software error has occurred. Contact your system support staff.

421 EPKI_NO_DEVICE Requested device not found
The device specified in the ioctl(2) request was not defined in the current configuration. Correct the device specified and then reissue the request.

422 EPKI_PROC_LIM The process limit per IOP device has been exceeded.
The open request was rejected because the number of processes with an IOP device open is at the packet driver limit.

When the packet driver is started, this limit is defined in the packet driver configuration file using the MAX_IOP_PROC parameter. If the limit is not specified in the configuration file, the IOP device process limit will default to 10.

Wait for a process to close an IOP device and then retry the open request.

The value may be changed after the packet driver has been started with command ipi3_option(8) or hpi3_option(8).

423 EPKI_ALREADY_ENBL The packet interface has already been enabled.
The packet interface has already been enabled. This is an informative message, and no action is required.

424 EPKI_DEV_CLEAR Device has been cleared
An ioctl(2) request was issued to a device that is in the process of being cleared or has been cleared. After a device has been cleared, no further ioctl(2) requests will be accepted until the device has been closed and reopened.

Close the device, reopen it, and then reissue the request.

425 EPKI_CHAN_DOWN Channel(s) to the device are down
The packet driver was unable to successfully complete a device clear request because the channel to the device is not in the correct state. Contact your system support staff.

426 EPKI_HALTIO_ERR Halt I/O request failed
The packet driver was unable to complete a device clear request because it could not terminate the outstanding IOP activity. Contact your system support staff.

427 EPKI_SELRST_ERR Selective reset request failed
The packet driver was unable to complete a device clear request because it could not successfully reset the device. Contact your system support staff.

428 EPKI_SETATTR_ERR Set attribute request failed
After a device was cleared, the packet driver was unable to reset the burst size for the device. Contact your system support staff.

429 EPKI_RESPBUF_LOST The contents of response buffer was lost.
Because the size of a packet exceeded the IOP maximum packet length, the IOP copied the command portion of the packet into a response buffer allocated by the packet driver.

The contents of this response buffer were lost.

This is an informative message, and no action is required.

430 EPKI_CMDLIST_LIM The command list limit per IPI-3 has been exceeded.
The number of command list requests sent by the user, but not yet received, is at the packet driver limit. This limit is defined when the packet driver is started, in the packet driver configuration file with parameter, MAX_STK_COUNT.

If the limit is not specified in the configuration file, the limit defaults to 5. This value may be changed after the packet driver has been started with the ipi3_option(8) or hpi3_option(8) command.

431 EPKI_DRIVER_DOWN The packet driver is down.
A packet driver device could not be opened because the packet driver has not been started. Wait until the packet driver has been started and then try again.

432 EPKI_PEND_SHUTDOWN The packet driver shutdown is pending.
A packet driver device could not be opened because a shutdown of the packet driver is pending. Wait until the packet driver has been restarted and then try again.

433 EPKI_DRIVER_UP The packet driver is up.
The packet driver is up. This is an informative message, and no action is required.

434 EPKI_PEND_STARTUP The packet driver startup is pending.
A request to start the packet driver was terminated because another start-up request is pending. This is an informative message, and no action is required.

435 EPKI_DRIVER_ACTIVE The packet driver is active.
A packet driver shutdown request failed because the packet driver is active. Wait until the packet driver is inactive and then try again.

436 EPKI_STOP_DRIVER The request to stop the IOP driver failed.
The packet driver could not successfully complete a shutdown request because it was unable to stop an IOP driver. Further information can be obtained from the pki_errno, pki_response, and pki_extsts fields in the shutdown response. Contact your system support staff.

437 EPKI_IOP_NOT_CONF The IOP is not configured.
An ioctl(2) request was issued to an IOP that is not configured, or an open(2) request was issued to a device configured on an IOP that has been shut down. Reconfigure the IOP by using the ipi3_start(8) command and reissue the request.

438 EPKI_IOP_SHUTDOWN The IOP has been shut down.
A request was terminated because the IOP processing the request has been shut down. Reconfigure the IOP by using the ipi3_start(8) command and reissue the request.

439 EPKI_ALREADY_CONFIG A single IOP could not be restarted because it is still configured.
: A configuration request was terminated because the hardware to be defined is already configured.

500 EFSEMANA (SFS) Fast lock not available
: A request to assign a fast lock to a shared file failed because no more hardware semaphores were available.

501 EFSNOGROW (SFS) File has allocation restrictions
: A request to change the allocation of a shared file failed because the file was previously set to a state where no allocation changes are allowed.

502 EFSNOTEXCL (SFS) File is not exclusive
: A nonblocking request to exclusively open a shared file failed because the file is already in an open state by some other process.

503 EFSEXCLWR (SFS) File is write protected
: A nonblocking request to obtain a read lock on a shared file failed because the file is currently write locked by some other process.

504 EFSESDOWN (SFS) External semaphore device unavailable
: An attempt to utilize the shared file system has failed because the external semaphore device is unavailable.

540 ENOTWELLFORMED I/O request not well formed
: An I/O request that is not wellformed has been issued against a file that was opened with the O_WELLFORMED option. An I/O request is considered wellformed only if the file offset is exactly on a sector boundary, the I/O request length is exactly a whole number of sectors, and the I/O buffer address in common memory is on a word boundary.

546 EFSBAD (Panicless File System) File system corrupted
: The panicless file system consistency checking code has detected an error in a file system super block or dynamic block. The file system has been marked in error, and must be repaired with /etc/fsck.

547 EDBAD (Panicless File System) Directory corrupted
: The panicless file system consistency checking code has detected an error in a directory entry. The entry has been marked in error, and must be repaired with /etc/fsck.

549 EIBAD (Panicless File System) File Inode Corrupted
: The panicless file system consistency checking code has detected an error in a file inode table entry. The file has been marked in error, and must be repaired with /etc/fsck.

SEE ALSO

`intro`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

UNICOS File Formats and Special Files Reference Manual, Cray Research publication SR–2014

Scientific Libraries Reference Manual, Cray Research publication SR–2081

Intrinsic Procedures Reference Manual, Cray Research publication SR–2138

UNICOS Macros and Opdefs Reference Manual, Cray Research publication SR–2403

NAME

accept – Accepts a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (int s, struct sockaddr *addr, int *addrlen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The accept system call accepts a connection on a socket. It accepts the following arguments:

s Specifies the descriptor for a socket. The socket was created by using socket(2), bound to an address with bind(2), and is listening for connections after a listen(2) request. The accept call extracts the first connection on the queue of pending connections, creates a new socket with the same properties as *s*, and allocates a new file descriptor for the socket. If no pending connections exist on the queue, and the socket is not marked as nonblocking, accept blocks the caller until a connection exists. If the socket is marked nonblocking, and no pending connections exist on the queue, accept returns an error as described in the RETURN VALUES subsection that follows. The accepted socket cannot be used to accept more connections. The original *s* socket remains open and listens for other connections.

addr Specifies the address of a sockaddr structure. When a request arrives, the accept system call fills this sockaddr structure with the address of the client that placed the request. The domain in which the communication is occurring determines the exact format of the *addr* argument.

addrlen Specifies the address of an integer. The accept system call fills this integer with the length of the address that was placed in the sockaddr structure pointed to by *addr*. Initially, it must contain the amount of space to which *addr* points; on return, it contains the actual number of bytes in the address that is returned.

The accept call sets up send and receive socket buffers (sockbufs) using the sockbuf space limit of the listening *s* socket. The accept call fails and returns an ELIMIT error if this call would cause the user's per-session sockbuf space limit to be exceeded. The original *s* socket remains open and continues to listen.

This call is used with connection-based socket types; it is currently used with SOCK_STREAM.

To determine whether a connection is ready to be accepted, instead of issuing the accept system call, you can issue the select(2) system call and set the bit for the socket file descriptor in the read mask (*readfds*).

For protocols that require an explicit confirmation, the `accept` call merely dequeues the next connection request; it does not imply confirmation. Confirmation can be implied by a standard read or write operation on the new file descriptor; rejection can be implied by closing the new socket.

You can obtain user connection request information without confirming the connection by issuing a `recvmsg`(2) call with a `msg_iovlen` value of 0 and a nonzero `msg_control` value, or by issuing a `getsockopt`(2) call. Similarly, you can provide information about user connection rejection by issuing a `sendmsg`(2) call and providing only the control information, or by issuing a `setsockopt`(2) call.

NOTES

If *addrlen* is less than the size of the address of the connecting entity (that is, less than the size of a `struct sockaddr`), the `accept` call truncates its result to fit into the available space.

If the `SOCKET_MAC` configuration option is enabled, the active security label of the process must be greater than or equal to the security label of the socket. The `SOCKET_MAC` configuration option is part of the TCP/IP configurable feature variables list in `uts/cf/Nmakefile`. For more information, see the `connect`(2) man page.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
`PRIV_MAC_READ`	The process is allowed to override the security level and compartment restrictions when the `SOCKET_MAC` configuration option is enabled.

RETURN VALUES

If `accept` completes successfully, it returns a nonnegative integer that is a descriptor for the accepted socket; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `accept` call fails if one of the following error conditions occurs:

Error Code	Description
`EACCES`	If the `SOCKET_MAC` configuration option is enabled, the process does not meet the security level and compartment requirements and does not have the appropriate privilege.
`EBADF`	Descriptor is invalid.
`EFAULT`	Argument *addr* or argument *addrlen* is not in the user address space in which it can be written.
`EINVAL`	Socket is not bound or socket is not listening.
`ELIMIT`	The user's socket buffer space limit is exceeded.
`ENOTSOCK`	Descriptor is not a socket.

EWOULDBLOCK Socket is marked nonblocking, and no connections are present to be accepted.

EXAMPLES

This server program shows how to use the `accept` system call in context with other TCP/IP calls. (Some system calls in this example are not supported on Cray MPP systems.) The program simply creates a TCP/IP socket, waits for a client process from some host to attempt a connection, accepts the connection, and forks a child process to provide the service to the client.

The original (parent) server loops back to look for additional connection attempts while the temporary (child) server reads a string of data sent by the client process.

```
/*  Server side of client-server socket example. For client side,
    see socket(2).
    Syntax: server portnumber & */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

main(int argc, char *argv[])
{
     int s, ns;
     struct sockaddr_in src;          /* source socket address */
     int len=sizeof(src);
     char buf[256];

     /* create port */
     src.sin_family = AF_INET;
     src.sin_port = atoi(argv[1]);
     src.sin_addr.s_addr = 0;

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("server, unable to open socket");
          exit(1);
     }

     while (bind(s, (struct sockaddr *) &src, sizeof(src)) < 0) {
          printf("Server waiting on bind...\n");
          sleep(1);
     }

     listen(s, 5);
```

```
        while (1) {
             ns = accept(s, (struct sockaddr *) &src, &len);
             if (ns < 0) {
                  perror("server, accept failed");
                  exit(1);
             }

             if (fork() == 0) {
                  /* in child server */
                  close(s);     /* child will use socket ns, parent uses s */
                  read(ns, &buf, sizeof(buf));
                  printf("Server read: %s\n", buf);
                  close(ns);
                  exit(0);
             }
             close(ns);         /* close socket used by child */
        }
   }
```

FILES

`/etc/config/spnet.conf`	Contains the network access list
`/usr/adm/sl/slogfile`	Receives security log records
`/usr/include/sys/socket.h`	Contains definitions related to sockets, types, address families, and options
`/usr/include/sys/types.h`	Contains types required by ANSI X3J11

SEE ALSO

`bind`(2), `connect`(2), `getsockopt`(2), `listen`(2), `select`(2), `sendmsg`(2), `setsockopt`(2), `socket`(2)

UNICOS File Formats and Special Files Reference Manual, Cray Research publication SR–2014

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

`access` – Determines accessibility of a file

SYNOPSIS

```
#include <unistd.h>
int access (const char *path, int amode);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `access` system call determines the accessibility of the path name pointed to by the *path* argument. It checks for the file access permissions indicated by *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

It accepts the following arguments:

path Points to a file path name.

amode Identifies the bit pattern to check against the file's bit pattern denoting access permission. Construct the bit pattern in *amode*, as follows:

`00` or `F_OK`	Check existence of file
`01` or `X_OK`	Execute (search)
`02` or `W_OK`	Write
`04` or `R_OK`	Read
`020` or `G_OK`	Check for `set-gid` bit
`040` or `U_OK`	Check for `set-uid` bit
`0400` or `EUID_OK`	Test using the effective IDs rather than the real IDs

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 permissions checked with respect to the group mode bits; and all others have permissions checked with respect to the other mode bits.

NOTES

If the file has an access control list, users that are not the file owner have permissions checked with respect to the access control list. Users who are not affected by entries in the access control list have permissions checked with respect to the other mode bits.

Permission to the file is also based on a comparison between the active security label of the process and the security label of the file. These comparisons are summarized as follows:

- For read, execute, or search permission, the active security label of the process must dominate the security label of the file.
- For write permission, the active security label of the process must equal the security label of the file.

If the file is a labeled device, the active security label of the process must fall within the security label range of the device.

Only appropriately authorized users are granted permission to files that are in the OFF state or that are multilevel.

The process must be granted search permission to every component of the path prefix via the permission bits and access control list.

The process must be granted search permission to every component of the path prefix via the security label.

If FSETID_RESTRICT is enabled, only a process with appropriate privileges can be granted write permission to set-user-ID or set-group-ID files.

A process with the effective privileges shown are granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_DAC_OVERRIDE	The process is granted read, execute, search, or write permission to the file via the permission bits and access control list.
PRIV_FSETID	If FSETID_RESTRICT is enabled, the process is granted write permission to the set-user-ID or set-group-ID file.
PRIV_MAC_READ	The process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_READ	The process is granted read, execute, or search permission to the file via the security label.
PRIV_MAC_WRITE	The process is granted write permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted read, execute, search, or write permission to the file. The super user or a process with the suidgid permission can override the restriction introduced by the FSETID_RESTRICT system configuration option.

RETURN VALUES

If `access` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `access` system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	Permission bits of the file mode do not permit the requested access.
EACCES	Search permission is denied on a component of the path prefix.
EACCES	The file is in the `OFF` state and the calling process does not have appropriate privileges.
EACCES	The file is a multilevel file and the calling process does not have appropriate privileges.
EACCES	The security label of the file does not allow the requested access.
EACCES	If the `FSETID_RESTRICT` and `PRIV_SU` configuration options are enabled, the process does not have appropriate privilege to gain write permission to the set-user-ID or set-group-ID file.
EFAULT	The *path* argument points outside the allocated process address space.
EMANDV	The security label range of a device does not allow the requested access.
ENAMETOOLONG	The *path* argument is longer than `PATH_MAX` characters.
ENOENT	The specified file does not exist.
ENOENT	Read, write, or execute (search) permission is requested for a null path name.
ENOTDIR	A component of the path prefix is not a directory.
EROFS	Write access is requested for a file on a read-only file system.
ETXTBSY	Text file is busy.

Mandatory access violations are recorded in the security log for these conditions:

Error Code	Description
EACCES	The user's security label does not allow access to the file.
EINTCATV	The user's active category does not match the file's category.
EMANDV	The caller is attempting to open an existing, labeled device but does not have an active security label that falls within the authorized security label range of the device.

FORTRAN EXTENSIONS

The access system call can be called from Fortran as a function (on all systems except Cray MPP systems and CRAY T90 series systems):

```
CHARACTER*n path
INTEGER amode, ACCESS, I
I = ACCESS (path, amode)
```

path may also be an integer variable. In this case, the data must be packed 8 characters per word and terminated with a null (0) byte. The PXFACCESS(3F) subroutine provides similar functionality and is available on all Cray Research systems.

EXAMPLES

This example illustrates how to use the access system call to check on the existence of a file before issuing an open(2) call. The access request verifies that file datafile exists before an attempt is made to open it.

```
int fd;

if (access("datafile", F_OK)) {
     fprintf(stderr, "File datafile does not exist.\n");
}
else {
     fd = open("datafile", O_RDONLY);
}
```

FILES

/usr/include/unistd.h	Contains C prototype for the access system call

SEE ALSO

chmod(2), stat(2)

PXFACCESS(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

acct – Enables or disables process accounting

SYNOPSIS

```
#include <unistd.h>
int acct (char *path);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The acct system call enables or disables the system process accounting routine. If the routine is enabled, an accounting record is written to an accounting file for each process that terminates. An exit(2) call or a signal can cause termination. Only a process with appropriate privilege can use this system call.

The acct system call accepts the following argument:

path Points to a path name that specifies the accounting file. The accounting file format is given in acct(5).

If *path* is nonzero and no errors occur during the system call, the accounting routine is enabled. If *path* is 0 and no errors occur during the system call, it is disabled.

If the accounting routine is already enabled and *path* differs from the accounting file currently in use, the accounting file will be switched to *path* without the loss of any accounting information.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_ACCT	The process is allowed to use this system call.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_ACCT permbit is allowed to use this system call.

RETURN VALUES

If acct completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The `acct` system call fails if one of the following error conditions occurs:

Error Code	Description
`EACCES`	A component of the path prefix denies search permission.
`EACCES`	The file specified by *path* is not a regular file.
`EACCES`	Write permission is denied for the specified accounting file.
`EFAULT`	The *path* argument points to an illegal address.
`EISDIR`	The specified file is a directory.
`ENOENT`	One or more components of the accounting file path name do not exist.
`ENOTDIR`	A component of the path prefix is not a directory.
`EPERM`	The process does not have appropriate privilege to use this system call.
`EROFS`	The specified file resides on a read-only file system.

FILES

`/usr/include/unistd.h`	Contains C prototype for the `acct` system call

SEE ALSO

`exit`(2)

`acct`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

acctctl – Checks status of, enables, and disables process, daemon, and record accounting

SYNOPSIS

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

int acctctl (int fnc, void *act);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The acctctl system call checks the status of, enables, and disables process, daemon, and record accounting. It accepts the following arguments:

fnc Identifies a function to be performed as follows:

Function	Description
AC_DMDAUTHORIZED	Returns an indication of whether the executing user is authorized to enable/disable accounting.
AC_DMDSTART	Enables the indicated accounting method.
AC_DMDSTOP	Disables the indicated accounting method.
AC_DMDCHECK	Checks the current state of a specific accounting method.
AC_DMDSTAT	Checks the current state of all accounting methods.

act Points to either an actctl, actstat, or actstat structure, depending on the function (i.e. *fnc*) to be performed.

This parameter is ignored for the AC_DMDAUTHORIZED function.

Enabling an accounting method requires an actctl structure which defines the daemon/record accounting identifier to be enabled (actctl.ac_stat.ac_id), the name of the file to write accounting data to (actctl.ac_path), and an optional parameter, which is defined by the accounting method being enabled (actctl.ac_stat.ac_param).

Disabling an accounting method requires an actctl structure which defines the daemon/record accounting identifier to be disabled (actctl.ac_stat.ac_id).

Checking the state of an accounting method requires an actstt structure which defines the daemon/record accounting identifier being checked (actstt.ac_id). On a successful return, the accounting method's current state (i.e. ACS_ON, or ACS_OFF) and optional parameter's value (actstt.ac_stt and actstt.ac_param, respectively) are set.

Checking the state of all accounting methods requires an `actstt` structure which defines the number of accounting status entries available (`ac_stat.ac_sttnum`). On a successful return, each available entry is filled in with the current state of an accounting method up to the number of accounting methods defined in the kernel. An accounting method's status entry can be accessed by using its defined identifier as the index into the `ac_stat`[] array returned.

NOTES

Only a process with appropriate privilege can use this system call to enable and/or disable accounting. However, no special privilege is required to check/status accounting states.

When enabling accounting, if the type of accounting specified is already enabled, the accounting file being used for data collection will be closed and switched to the file specified without losing any accounting information.

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

To be granted write permission to the file, the active security label of the process must equal the security label of the file.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
`PRIV_ACCT`	The process is allowed to use this system call to enable/disable accounting.
`PRIV_DAC_OVERRIDE`	The process is granted search permission to every component of the path prefix through the permission bits and access control list.
`PRIV_DAC_OVERRIDE`	The process is granted write permission to the file through the permission bits and access control list.
`PRIV_MAC_READ`	The calling process is granted search permission to every component of the path prefix through the security label.
`PRIV_MAC_WRITE`	The process is granted write permission to the file through the security label.

If the `PRIV_SU` configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted write permission to the file. The super user is allowed to use this system call to enable/disable accounting.

RETURN VALUES

If `acctctl` completes successfully, a value of 0 is returned and the structure pointed to by *act* is filled in as indicated above. If `acctctl` completes unsuccessfully, a value of −1 is returned, the structure pointed to be *act* is not modified and `errno` is set to indicate the error.

For the `AC_DMDAUTHORIZED` function, a value of 0 is returned if the executing user is authorized to enable/disable accounting. A value of −1 is returned and `errno` is set to `EPERM`, if the executing user is not authorized to enable/disable accounting.

ERRORS

The `acctctl` system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied on a component of the path prefix.
EACCES	The process is not granted write permission to the file.
EACCES	The file specified by *path* is not an ordinary file.
EFAULT	The *act* argument points to an illegal address.
EINVAL	An invalid argument was specified.
EISDIR	The file specified is a directory.
ENOENT	A component of the path prefix or the file does not exist.
EPERM	The process does not have appropriate privilege to use this system call.
EROFS	The specified file resides on a read-only file system.

FILES

`/usr/include/acct/dacct.h`	Defines daemon accounting files
`/usr/include/sys/acct.h`	Defines the `actctl`, `actstt`, and `ac_stat` structures
`/usr/include/sys/accthdr.h`	Defines daemon/record identifiers

SEE ALSO

`acct`(2), `dacct`(2)

`acct`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`acct`(8), `csaswitch`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

acctid – Changes account ID of a process

SYNOPSIS

```
#include <unistd.h>
int acctid (int pid, int acid);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The acctid system call changes the account ID of the specified process. Only a process with appropriate privilege can set the account ID.

The acctid system call accepts the following arguments:

pid Specifies the *pid* of the target process. A *pid* of 0 means the current process.

acid Specifies the value of the new account ID. The value must be either nonnegative or –1. An *acid* of –1 means no change.

For more information about changing the account ID of a user, see newacct(1).

NOTES

The active security label of the calling process must be greater than or equal to the active security label of the specified process.

To set the account ID of a process, the active security label of the calling process must be equal to the active security label of the specified process.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_ACCT	The calling process is allowed to set the account ID.
PRIV_MAC_READ	The calling process is allowed to override the restriction that its active security label must be greater than or equal to the security label of the specified process.
PRIV_MAC_WRITE	The calling process is allowed to override the security label restriction when setting the account ID of a process.
PRIV_POWNER	The calling process is considered the owner of the specified process.

If the PRIV_SU configuration option is enabled, the super user is considered the owner of the specified process and is allowed to set the account ID. The super user is allowed to override all security label restrictions.

RETURN VALUES

If `acctid` completes successfully, it returns the previous account ID of the specified process; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

ERRORS

The `acctid` system call fails if one of the following error conditions occurs:

Error Code	Description
`EINVAL`	One of the arguments contains an invalid value.
`EPERM`	The calling process does not have appropriate privilege to set the account ID.
`EPERM`	The calling process does not have appropriate privilege to use this system call.
`ESRCH`	The specified process could not be found.
`ESRCH`	The caller does not own the specified process and does not have appropriate privilege.
`ESRCH`	The calling process does not meet the security label requirements and does not have appropriate privilege.

FILES

`/usr/include/unistd.h`	Contains C prototype for the `acctid` system call

SEE ALSO

`newacct`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

adjtime – Corrects the time to allow synchronization of the system clock

SYNOPSIS

```
#include <sys/time.h>
int adjtime (struct timeval *delta, struct timeval *olddelta);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The adjtime system call adjusts the system's notion of the current time, as returned by gettimeofday(2). It advances or retards it by the amount of time specified in the struct timeval (defined in header file sys/time.h) pointed to by *delta*.

The adjtime system call accepts the following arguments:

delta Points to a timeval structure.

olddelta Points to a structure that contains, upon return, the time still to be corrected from the earlier call. If *olddelta* is a null pointer, the corresponding information is not returned.

The adjustment is effected by speeding up (if that amount of time is positive) or slowing down (if that amount of time is negative) the system clock by some small percentage, generally a fraction of 1%. Thus, the time is always a monotonically increasing function. A time correction from an earlier call to adjtime may not be finished when adjtime is called again.

This call is used in time servers that synchronize the clocks of computers in a local area network. Such time servers slow down the clocks of some machines and speed up the clocks of others to bring them to the notion of network time.

Only a process with appropriate privilege can use this system call.

The adjustment value is silently rounded to the resolution of the system clock.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_TIME	The process is allowed to use this system call.

If the PRIV_SU configuration option is enabled, the super user is allowed to use this system call.

RETURN VALUES

If `adjtime` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `adjtime` system call fails if one of the following error conditions occurs:

Error Code	Description
`EFAULT`	*olddelta* points to a region of the process' allocated address space that is not writable.
`EPERM`	The process does not have appropriate privilege to use this system call.

SEE ALSO

`gettimeofday`(2)

`date`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

alarm, _lwp_alarm – Sets a process alarm clock

SYNOPSIS

```
#include <unistd.h>
unsigned int alarm (unsigned int sec);
unsigned int _lwp_alarm (unsigned int sec);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The alarm system call instructs the alarm clock of the calling process to send the SIGALRM signal to the calling process after a specified number of real-time seconds has elapsed; see signal(2).

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

The alarm and _lwp_alarm system calls accept the following argument:

sec Specifies the number of real-time seconds. To cancel a previous alarm request, set *sec* to 0.

On Cray MPP systems, the alarm system call sets an alarm only for the processing element (PE) on which it is called. It has no effect on any other PE of the application.

The _lwp_alarm system call ensures that the alarm signal is sent to the specific member of the multitasking group that called _lwp_alarm. In contrast, alarm results in an alarm signal being sent to an arbitrary thread.

NOTES

The _lwp_alarm system call provides compatibility with the behavior of alarm previous to UNICOS 9.0. It is a transitional tool since it may disappear in a future release of the UNICOS operating system. This compatibility issue affects only multitasked applications.

RETURN VALUES

The alarm system call returns the amount of time previously remaining in the alarm clock of the calling process.

FORTRAN EXTENSIONS

The `alarm` system call can be called from Fortran as a function:

```
INTEGER sec, ALARM, I
I = ALARM (sec)
```

Alternatively, `alarm` can be called from Fortran as a subroutine. In this case, the return value of the system call is unavailable.

```
INTEGER sec
CALL ALARM (sec)
```

The Fortran program must not specify both the subroutine call and the function reference to `alarm` from the same procedure.

EXAMPLES

This example shows how to use the `alarm` request to notify the invoking process when a specific amount of time has expired. After the specified time (10 seconds), the `SIGALRM` signal is sent to the process, interrupting the process.

Using the `signal`(2) system call, the program requests that the function `handlr` be entered when the `SIGALRM` signal is received; otherwise, the process will usually terminate upon the receipt of the signal. After the function `handlr` executes, control is returned to the point of interruption.

```
#include <signal.h>
#include <unistd.h>

main()
{
     void handlr(int signo);

     signal(SIGALRM, handlr);
     alarm(10);

     /* After executing 10 seconds, SIGALRM signal interrupts program. */
     /* Execution resumes here after processing SIGALRM signal. */
}

void handlr(int signo)
{
     signal(signo, handlr);
     /* Process SIGALRM signal here and then return. */
}
```

FILES

`/usr/include/unistd.h`	Contains C prototype for the `alarm` system call

SEE ALSO

`pause`(2), `sigctl`(2), `signal`(2), `sigset`(2)

NAME

bind – Binds a name to a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int s, struct sockaddr *name, int namelen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The bind system call assigns a name to an unnamed socket. It accepts the following arguments:

s Specifies the descriptor of the socket to be bound.

name Points to the address of the sockaddr structure that contains the local address to which the socket should be bound.

namelen Specifies length of the address, pointed to by *name*. The length is measured in bytes.

When a socket is created by using socket(2), it is assigned a descriptor *s* and exists in a name space (address family), but it has no name assigned. The bind call requests that the name *name* be assigned to the socket.

The rules used in name binding vary among communication domains.

NOTES

The active security label of the process must equal the security label of the socket.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_MAC_WRITE	The process is allowed to override the security label restrictions.

If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions.

RETURN VALUES

If bind completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The `bind` system call fails if one of the following error conditions occurs:

Error Code	Description
`EACCES`	The process does not meet the security label requirements, and does not have appropriate privilege.
`EADDRINUSE`	Specified address is already in use.
`EADDRNOTAVAIL`	Specified address is not available from the local machine.
`EBADF`	Descriptor *s* is not valid.
`EFAULT`	Argument *name* is not in a valid part of the user address space.
`EINVAL`	Socket is already bound to an address or *namelen* is not the size of a valid address for the specified address family.
`ENOTSOCK`	Descriptor *s* is not a socket.

See `open`(2) for file system-related errors.

EXAMPLES

This server program shows how to use the `bind` system call in context with other TCP/IP calls. (Some system calls in this example are not supported on Cray MPP systems.) The program simply creates a TCP/IP socket, waits for a client process from some host to attempt a connection, accepts the connection, and forks a child process to provide the requested service to the client.

The original (parent) server loops back to look for additional connection attempts while the temporary (child) server reads a string of data that the client process sends.

```
/*  Server side of client-server socket example. For client side,
    see socket(2).
    Syntax: server portnumber & */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

main(int argc, char *argv[])
{
     int s, ns;
     struct sockaddr_in src;          /* source socket address */
     int len=sizeof(src);
     char buf[256];

     /* create port */
     src.sin_family = AF_INET;
     src.sin_port = atoi(argv[1]);
     src.sin_addr.s_addr = 0;

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("server, unable to open socket");
          exit(1);
     }

     while (bind(s, (struct sockaddr *) &src, sizeof(src)) < 0) {
          printf("Server waiting on bind...\n");
          sleep(1);
     }
```

```
        listen(s, 5);

        while (1) {
             ns = accept(s, (struct sockaddr *) &src, &len);
             if (ns < 0) {
                  perror("server, accept failed");
                  exit(1);
             }

             if (fork() == 0) {
                  /* in child server */
                  close(s);     /* child will use socket ns, parent uses s */
                  read(ns, &buf, sizeof(buf));
                  printf("Server read: %s\n", buf);
                  close(ns);
                  exit(0);
             }
             close(ns);         /* close socket used by child */
        }
   }
```

FILES

`/usr/include/sys/socket.h`	Header file for sockets
`/usr/include/sys/types.h`	Header file for types

SEE ALSO

`connect`(2), `getsockname`(2), `listen`(2), `open`(2), `socket`(2), `unlink`(2)

`inet`(4P), `intro`(4) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

brk, sbrk, sbreak – Changes data segment space allocation

SYNOPSIS

```
#include <unistd.h>
int brk (char *endds);
char *sbrk (int incr);
long *sbreak (int incr);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The brk, sbrk, and sbreak system calls dynamically change the amount of space allocated for the data segment of the calling process; see exec(2). The change is made by resetting the break value of the process and allocating the appropriate space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases. The newly allocated space is initialized to 0.

The brk, sbrk, and sbreak system calls accept the following arguments:

endds Specifies break value to be set (brk only). To specify the return value of the ulimit(2) system call as the value for *endds*, you must convert the number returned by ulimit(2) to a character pointer, as follows:

```
brk(((char *)0) + ulimit(3,0));
```

incr Specifies the number of bytes (sbrk) or words (sbreak) to add to the break value. To decrease the allocated space, specify *incr* as a negative number.

CAUTIONS

The use of the brk, sbrk, or sbreak system call directly interferes with the processing of library routine malloc(3C), which the calling sequence uses to implement run-time stack space.

RETURN VALUES

If brk completes successfully, a value of 0 is returned; sbreak and sbrk return the old break value. Otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The brk, sbrk, or sbreak system call fails without making any change in the allocated space if one of the following error condition occurs:

Error Code	Description
ENOMEM	More space is specified in the argument than is allowed by a system-imposed maximum (see limit(2) and ulimit(2)).
EMEMLIM	More memory space was requested than is allowed for the processes attached to this lnode. The maximum value is set by the -c option of the shradmin(8) command. This error appears only on systems running the fair-share scheduler.

FORTRAN EXTENSIONS

The brk system call can be called from Fortran as a function:

```
INTEGER endds, BRK, I
I = BRK (endds)
```

Alternatively, brk can be called from Fortran as a subroutine. In this case, the return value of the system call is unavailable.

```
INTEGER endds
CALL BRK (endds)
```

The Fortran program must not specify both the subroutine call and the function reference to brk from the same procedure.

The sbrk system call can be called from Fortran as a function:

```
INTEGER incr, SBRK, I
I = SBRK (incr)
```

The sbreak system call can be called from Fortran as a function:

```
INTEGER incr, SBREAK, I
I = SBREAK (incr)
```

EXAMPLES

The following examples illustrate how to use the brk, sbrk, and sbreak system calls to expand and decrease the size of the calling process. The first three examples, involving the same expansion task, highlight differences in these calls.

Example 1: This brk request expands the size of the calling process by 1000 octal words (8 bytes per word).

A sbrk request first determines the current break value for the process from which the new break value is calculated. Then, brk expands the calling process size.

```
char *brkp;
int rtrn;

brkp = sbrk(0);      /* return current break value for process. */
rtrn = brk(brkp + (8 * 01000));
```

Example 2: The sbrk request expands the size of the calling process by 1000 octal words (8 bytes per word). The return value placed in ptr points to the beginning of the newly allocated block of 1000 octal words.

```
char *ptr;

ptr = sbrk(8 * 01000);
```

Example 3: The sbreak request expands the size of the calling process by 1000 octal words (8 bytes per word). The return value placed in ptr points to the beginning of the newly allocated block of 1000 octal words.

```
long *ptr;

ptr = sbreak(01000);
```

Example 4: This sbreak request decreases the size of the calling process by 2000 octal words (8 bytes per word):

```
long *ptr;

ptr = sbreak(-02000);
```

Example 5: The following brk system call increases the size of the calling process to the maximum allowed for the user's process:

```
brk(((char *)0) + ulimit(3,0));
```

FILES

/usr/include/unistd.h	Contains C prototype for the brk, sbrk, and sbreak system calls

SEE ALSO

exec(2), limit(2), ulimit(2)

malloc(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

shradmin(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`chacid` – Changes disk file account ID

SYNOPSIS

```
#include <unistd.h>
int chacid (char *path, int acid);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `chacid` system call changes the account ID associated with a disk file. (The file can be the original file or a symbolic link.) The `chacid` system call accepts the following arguments:

path Specifies the path name of the file to be changed.

acid Specifies the account ID or –1. If *acid* is –1, the current account ID is returned, and no change is made.

All users may call `chacid` with an *acid* of –1; however, only a process with appropriate privilege may call `chacid` with an account ID other than –1.

NOTES

The active security label of the calling process must be greater than or equal to the active security label of the file.

To set the account ID of a file, the active security label of the calling process must equal the active security label of the file.

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
`PRIV_ACCT`	The calling process is allowed to set the account ID of the file.
`PRIV_DAC_OVERRIDE`	The process is granted search permission to a component of the path via the permission bits and access control list.
`PRIV_MAC_READ`	The calling process is allowed to override the restriction that its active security label must be greater than or equal to the security label of the file.
`PRIV_MAC_READ`	The process is granted search permission to a component of the path via the security label.

PRIV_MAC_WRITE — The calling process is allowed to override the security label restriction when setting a file's account ID.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix. The super user is allowed to set the account ID of a file. If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions.

RETURN VALUES

If chacid completes successfully, it returns the previous account ID associated with the file; otherwise, a value of –1 is returned, and errno is set to indicate the error. If the *acid* argument is –1, the return value is the current account ID associated with the file.

ERRORS

The chacid system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	A component of the path prefix denies search permission.
EINVAL	The file is not on a local file system.
EINVAL	The calling process does not meet security label requirements and does not have appropriate privilege.
EINVAL	The *acid* argument contains an invalid value.
EPERM	The calling process does not have appropriate privilege to set the file account ID.
EQACT	A file or inode quota limit was reached for the current account ID.

FILES

/usr/include/unistd.h — Contains C prototype for the chacid system call

SEE ALSO

acctid(2)

chacid(1), newacct(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

chdir – Changes working directory

SYNOPSIS

```
#include <unistd.h>
int chdir (const char *path);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The chdir system call causes a specified directory to become the current working directory; that is, the starting point for path searches for path names not beginning with /. The chdir system call accepts the following argument:

path Points to the directory path name.

NOTES

To be granted search permission to a component of the path name, the active security label of the process must be greater than or equal to the security label of the component.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to a component of the path via the permission bits and access control list.
PRIV_MAC_READ	The process is granted search permission to a component of the path via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path name.

RETURN VALUES

If chdir completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The chdir system call fails and the current working directory remains unchanged if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied for any component of the path name.
EFAULT	The *path* argument points outside the allocated process address space.
ENOENT	The specified directory does not exist.
ENOTDIR	A component of the path name is not a directory.

FORTRAN EXTENSIONS

The chdir system call can be called from Fortran as a function (on all systems except Cray MPP systems and CRAY T90 series systems):

```
CHARACTER*n path
INTEGER CHDIR, I
I = CHDIR (path)
```

Alternatively, chdir can be called from Fortran as a subroutine (on all systems except Cray MPP systems and CRAY T90 series systems). In this case, the return value of the system call is unavailable.

```
CHARACTER*n path
CALL CHDIR (path)
```

The Fortran program cannot specify both the subroutine call and the function reference to chdir from the same procedure. *path* may also be an integer variable. In this case, the data must be packed 8 characters per word and terminated with a null (0) byte. The PXFCHDIR(3F) subroutine provides similar functionality and is available on all Cray Research systems.

EXAMPLES

The following chdir request changes the current working directory in the invoking process environment to the parent directory of the current working directory:

```
if (chdir("..")) {
     fprintf(stderr, "The directory change was unsuccessful.\n");
     exit(1);
}
```

FILES

`/usr/include/unistd.h`	Contains C prototype for the `chdir` system call

SEE ALSO

`chroot`(2)

`PXFCHDIR`(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

chdiri – Changes a directory by using the inode number

SYNOPSIS

```
int chdiri (long dev, long ino, long gen);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The chdiri system call provides the user with a pathless change-directory operation on native UNICOS file systems. It locates a directory by using the inode number, and then it causes this directory to become the current directory.

The chdiri system call accepts the following arguments:

dev Specifies the device number. This number is built by the makedev macro that is defined outside of the kernel.

ino Specifies an inode number for the directory as reported by the ls –i command.

gen Specifies the generation number of the inode.

This provides a unique identification for a specific file. The generation number changes when an inode is reused. To print inode generation values, use the fck(1) command with the i and l options.

NOTES

Only a process with appropriate privilege can use this system call.

If the PRIV_SU configuration option is enabled, the super user is allowed to use this system call.

A process with the PRIV_MAC_READ and PRIV_DAC_OVERRIDE effective privileges is allowed to use this system call. See the effective privilege discussion in the NOTES section of the chdir(2) man page for additional privilege requirements. The chdir(2) search access discussions do not apply to this system call.

RETURN VALUES

If chdiri completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The chdiri system call fails and the current working directory remains unchanged if one of the error conditions listed on the chdir(2) man page occurs.

FILES

`/usr/include/sys/sysmacros.h`	Contains a description of the `makedev` macro

SEE ALSO

`chdir`(2)

`fck`(1), `ls`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

General UNICOS System Administration, Cray Research publication SG–2301

NAME

chkpnt – Checkpoints a process, multitask group, or job

SYNOPSIS

```
#include <sys/category.h>
#include <sys/restart.h>

int chkpnt (int category, int id, char *path, long flags);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The chkpnt system call creates a file containing all the information needed to restore the target processes identified by *category* and *id* to their saved execution state by the restart(2) system call. The file created is referred to as a *restart file*.

The chkpnt system call accepts the following arguments:

category Specifies C_PROC for a process or C_SESS for a job (or interactive session).

id Specifies the *pid* or *jid* corresponding to *category*. *id* = 0 assumes current process or current job, respectively.

path Specifies the path name of the restart file to be created.

flags Identifies optional actions.

The flags present in this field are OR'ed together to define the optional action to be performed by chkpnt. Currently, the only defined flag value is CHKPNT_KILL, which causes the target processes to die after the recovery image is complete.

By default, the restart file is protected from user modification and can be read only by the owner (file mode 0400) unless one of the following conditions is true:

- One of the processes in the chkpnt collection has an effective user ID (UID) different from the effective UID of the process performing chkpnt.
- One of the processes in the chkpnt collection has a security label that is different from the security label of the process performing chkpnt.
- One of the processes in the chkpnt collection has one of the following flags set:
  - PC_NOCORE (The process does not have read access to its executable image.)
  - PC_SECCORE (The process is permitted to use privilege and the SECURE_MAC configuration option is enabled.)
- One of the processes in the chkpnt collection is a setuid application.

The restart file is recognized by the system as a restart file because the type field of the restart file inode identifies the file as a regular file and the S_IRESTART (restart file attribute) bit is also set (see /usr/include/sys/stat.h).

Whenever any process is selected to be included in a restart file, all of its multitask group sibling processes are also included, because the meaningful recovery of any process requires that all of its multitask group siblings also be restored on recovery.

Processes with open pipes can be checkpointed and restarted if their pipe connections do not go outside the job or multitask group being checkpointed. To checkpoint a process with open pipes, all of its pipe connections must terminate with processes that are also to be included in the restart file.

Processes with open files that reside on network file system (NFS) file systems can be checkpointed and restarted. To restart a process with open NFS files, the NFS file systems on which the files reside have to be mounted unless the NFS file systems are managed by the automounter. In this case, the automounter will try to remount the file systems automatically.

Processes with open files that reside on Distributed File System (DFS) file systems can be checkpointed and restarted. The following conditions must exist in order for a user to restart a process with an open DFS file.

- The DFS client must be running on the local host.
- The DFS server must be running on the host where the file resides.

Access to DFS files is controlled by the user's Distributed Computing Environment (DCE) credentials as opposed to user identification (UID) and group identification (GID). DFS credentials consist of Kerberos tickets stored in a special file. When a process is checkpointed, a reference to these credentials is stored in the restart file. The credentials must still be present and valid when restart(2) is performed. If the credentials are no longer present or have expired, accesses to DFS files that are performed after the restart(2) system call will appear to be from the UID –2.

If the chkpnt system call writes the restart file into a directory that is being accessed via DFS, that directory must reside on a Cray Research DFS server. DFS servers of other manufacturers do not support the restart file type; if chkpnt tries to write a restart file into one of these directories, the call fails without returning an error.

Processes with unlinked files can be checkpointed and restarted if the total size of all unlinked files in use by the target process set is within the size limit established by the system administrator. See the MAX_UNLINKED_BYTES system variable in the /usr/src/uts/c1/cf/config.h file to see the site local definition.

With SSD solid-state storage devices, processes that are using secondary data segments (SDS) can be checkpointed and restarted if sufficient disk space is available to contain an image of the process SDS area within the restart file. An ENOSDS error may occur at restart time if the SDS area available at that time is less than what was in use at checkpoint time. The ENOSDS error means that restart(2) must be retried at a later time when sufficient SDS space is available.

Processes using online tape files cannot be checkpointed or restarted.

NOTES

The following restrictions apply to processes and jobs (including interactive sessions) that are to be checkpointed:

- Only a process with appropriate privilege may checkpoint or restart another user's job or process.
- The active security label of the job of the calling process must dominate the security label of the job or process being checkpointed, unless the caller has appropriate privilege.
- Processes with open pipes may be checkpointed and restarted successfully if the following two conditions are met:
  - All openings of the pipe file must be contained within the process collection being checkpointed.
  - All I/O operations on the pipe must be atomic with respect to the `chkpnt` system call. This condition is a limit on the size of an I/O operation: either `PIPE_BUF` bytes, or (`v_maxpipe * 4096`) bytes. `PIPE_BUF` is found in the `sys/param.h` file. `v_maxpipe` is a member of the `var` structure in the `sys/var.h` file.
- All files that a process was using when it was checkpointed must be present when the process is restarted. These files include all open files, any shared-text executables that the process was using such as shells, and the present working directory. In the restart file, each of these files is identified by its inode number and the minor number of the file system. If either changes, the `restart`(2) system call fails, and the call returns an `EFILERM` error. For example, if a file system is restored by `/etc/restore`, any process that was using files on that file system and that was checkpointed before the restore, will fail to restart. After the restore, each file on the file system has a different inode number than it did when the process was checkpointed.
- Processes using online tapes cannot be checkpointed or restarted.
- Processes using shared memory segments (CRAY T90 series systems only) cannot be checkpointed or restarted.

A process with the effective privilege shown is granted the following abilities:

Privilege	Description
`PRIV_DAC_OVERRIDE`	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
`PRIV_DAC_OVERRIDE`	The process is granted write permission to the directory containing the new restart file via the permission bits and access control list.
`PRIV_MAC_READ`	The process is granted search permission to every component of the path prefix via the security label.
`PRIV_MAC_WRITE`	The active security label of the calling process is considered equal to the active security label of every process being checkpointed.

PRIV_MAC_WRITE	The process is granted write permission to the directory containing the new restart file via the security label.
PRIV_POWNER	The calling process is considered the owner of every process being checkpointed.

If the PRIV_SU configuration option is enabled, the super user is considered the owner of every process being checkpointed. The super user is granted all access necessary to create the new restart file.

RETURN VALUES

If chkpnt completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The chkpnt system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied on a component of the restart file path prefix.
EACCES	The directory in which the restart file is to be created does not permit writing.
EAGAIN	One of the processes being checkpointed was never in a state that allowed checkpointing during the last 120 seconds. The chkpnt system call may be attempted again.
EEXIST	A file by the name of *path* already exists.
EEXIST	An open NFS file descriptor that is unlinked cannot be saved in the restart file.
EFAULT	The *path* argument points outside the allocated process address space.
EFBIG	To complete the checkpoint operation, the restart file would have to be larger than the file size limit of the calling process or the maximum file size.
EFILERM	One or more of the target processes has one or more unlinked files open, and the sum of the sizes of all unlinked files open by the target processes exceeds the site-configured, unlinked file, contents recovery limit.
EFILESH	One or more of the target processes has an open, unlinked regular file that is also open by one or more processes outside the target process set.
EFOREIGNFS	An operation that is supported only on local file systems was attempted on a nonlocal (foreign) file system.
EINVAL	An invalid argument was passed to the system call.
EINVAL	The caller has specified that a restart file of an entire job be created, and one or more of the processes in the job is in a process group whose process group leader is outside the job.
ENFILE	The system inode table is full.

ENOENT	A component of the restart file path prefix does not exist.
ENOENT	The restart file path name is null.
ENOSPC	Insufficient file space is available to create the restart file.
ENOTBLK	An unrecoverable resource is associated with the target process set.
ENOTDIR	A component of the restart file path prefix is not a directory.
ENOTTY	One or more of the processes has an unrecoverable character device open.
ENOTTY	The caller has specified that a restart file of an entire job be created, and two or more processes in the job have different controlling ttys.
EPERM	The caller was not running as root and specified a *pid* for which the caller's real or effective *uid* differed from the real or effective user ID of the target process.
EPERM	The caller did not have appropriate privilege and specified *category* as C_SESS.
EPERM	The active security label of the caller did not dominate that of the job or process being checkpointed.
EPIPE	One or more of the target processes has an open pipe that goes outside the target process set.
EQACT	A file or inode quota limit was reached for the current account ID.
EQGRP	A file or inode quota limit was reached for the current user ID.
EQUSR	A file or inode quota limit was reached for the current group ID.
EROFS	The *category* argument is C_PROC and no process exists with the requested process ID.
ESHMA	The process has a shared memory segment or segments attached (CRAY T90 series systems only), and cannot be checkpointed.
ESOCKTNOSUPPORT	Either support for the specified socket type has not been configured into the system, or no implementation for it exists. Check the protocol argument on the system call.
ESRCH	The specified restart file would reside on a read-only file system.
EXDEV	The specified restart file would reside on a foreign file system (for example, a remote file accessed with NFS).
EXDEV	One or more of the target processes has a file open on a foreign file system (for example, a remote file accessed with NFS).
EXDEV	One or more of the target processes has another process open through the */proc* file system and that process is not included in the target process set. A file or inode quota limit was reached for the current group ID.

EXAMPLES

The following examples illustrate different uses of the chkpnt system call.

Example 1: The chkpnt system call produces a checkpoint file (named chkpnt.pid) of the invoking process in the current working directory:

```
char filename[256], pid_char[8];
int pid;

pid = getpid();              /* get pid of current process */
sprintf(pid_char, "%d", pid);/* convert pid to char format */
strcpy(filename, "chkpnt."); /* create filename for chkpnt with */
strcat(filename, pid_char);  /* format chkpnt.pid */

if (chkpnt(C_PROC, pid, filename, 0) != 0) {
     perror("chkpnt failed");
}
```

Example 2: The chkpnt system call produces a checkpoint file (named chkpnt.jid) of the job containing the invoking process in the current working directory. After the checkpoint file is created, the job is immediately terminated.

```
char filename[256], jid_char[8];
struct jtab jdata;
int jid;

jid = getjtab(&jdata);       /* get jid of current job */
sprintf(jid_char, "%d", jid);/* convert jid to char format */
strcpy(filename, "chkpnt."); /* create filename for chkpnt with */
strcat(filename, jid_char);  /* format chkpnt.jid */

if (chkpnt(C_SESS, jid, filename, CHKPNT_KILL) != 0) {
     perror("chkpnt failed");
}
```

SEE ALSO

chmod(2), chown(2), creat(2), getpid(2), mknod(2), open(2), pipe(2), restart(2), setuid(2)

chkpnt_util(1), chkptint(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

chmem – Retrieves or modifies system physical memory availability

SYNOPSIS

```
#include <unistd.h>
#include <sys/map.h>
int chmem (long request, long *result);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The chmem system call provides a mechanism for determining the physical memory available to the host system and, for appropriately privileged processes, the capability to modify how much physical memory is available.

The chmem system call has the following arguments:

request Specifies the amount (in words) by which the system's notion of physical memory will be changed. All *requests* are rounded up to the nearest 512-word size. To retrieve the current notion of system physical memory, specify 0 as the value of *request*.

result Specifies an address.

The chmem system call returns the system's current notion of physical memory after the requested change has been considered, at the address specified by *result*. If *result* is null, no data is returned.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_RESOURCE	The process is allowed to modify system physical memory availability.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_SYSPARAM permbit is allowed to modify system physical memory availability.

RETURN VALUES

If chmem completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The chmem system call fails if one of the following error conditions occurs:

Error Code	Description
EAGAIN	An attempt to reduce the system's notion of physical memory could not be satisfied, probably because an unmovable process was locked into the portion of physical memory being taken down. Retry the procedure later.
EINVAL	The *result* address supplied was invalid.
ENOSPC	An attempt to increase the system's notion of physical memory would expand beyond the compile-time configured maximum amount of memory.
ENXIO	When the call was manipulating a bit map of memory, an error occurred. The chmem interface was disabled, returning –1 to all subsequent *requests*.
EPERM	A nonzero *request* was made by a process without appropriate privilege.

FILES

/usr/include/sys/types.h	Contains types required by ANSI X3J11
/usr/include/unistd.h	Contains C prototype for the chmem system call

NAME

chmod, fchmod – Changes the mode of a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod (const char *path, mode_t mode);
int fchmod (int fildes, mode_t mode);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4 (applies only to chmod)

DESCRIPTION

The chmod and fchmod system calls set the access permission portion of the specified file's mode as specified by the following arguments:

path Points to a file path name.

mode Specifies the bit pattern denoting the file's access permission. (See the header file, sys/stat.h, for a description of these bits.)

fildes Specifies the file descriptor.

To set the mode of a file, the process must be the file owner or have appropriate privilege. To set the mode of a restart file, the process must have appropriate privilege.

If the process is not a member of the file's owning group and the process does not have appropriate privilege, then the file S_ISGID mode bit (set group ID on execution) is cleared.

If the S_ISGID bit (set group ID on execution) is set and the S_IXGRP bit (execute or search by group) is not set, mandatory file or record locking will exist on a regular file. This can affect subsequent calls to creat(2), listio(2), open(2), read(2), reada(2), trunc(2), write(2), and writea(2) on this file.

If the S_ISVTX (sticky) bit is set on a directory, only the directory owner or a process with appropriate privilege can delete or rename files in that directory.

NOTES

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

The process must be granted write permission to the file via the security label. That is, the active security label of the process must equal the security label of the file.

If the FSETID_RESTRICT system configuration option is enabled, only a process with appropriate privilege can set the set-user-ID or set-group-ID mode bits. If a process does not have appropriate privilege, the set-user-ID and set-group-ID mode bits are cleared.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_FOWNER	The process is considered the file owner.
PRIV_FSETID	If the FSETID_RESTRICT system configuration option is enabled, the process is allowed to set the set-user-ID mode bits.
PRIV_MAC_READ	The process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_WRITE	The calling process is granted write permission to the file via the security label.
PRIV_RESTART	The process is allowed to set the mode of a restart file.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix. The super user is allowed to set the mode of a restart file. The super user is considered the file owner.

If the PRIV_SU configuration option is enabled, the super user is allowed to set the set-user-ID and set-user-ID mode bits and is granted write permission to the file via the security label.

RETURN VALUES

If chmod or fchmod completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and errno is set to indicate the error.

ERRORS

The chmod or fchmod system call fails and the file mode remains unchanged if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The *path* argument points outside the allocated process address space.
ENOENT	The specified file does not exist.
ENOTDIR	A component of the path prefix is not a directory.
EROFS	The specified file resides on a read-only file system.

EMANDV	User's compartments and level are not equal to that of the file.
EMANDV	The calling process does not have MAC write access to the file to which the file descriptor refers.
EMANDV	The process is not granted write permission to the file via the security label.
EPERM	The process is not the file owner and does not have appropriate privilege.
EPERM	The file is a restart file, and the process does not have appropriate privilege.
EPERMIT	User does not have permission to set or change the mode of a file to setuid or setgid.
EPERMIT	If the FSETID_RESTRICT system configuration is enabled, the process does not have appropriate privilege to set the set-user-ID or set-group-ID mode bits.

FORTRAN EXTENSIONS

The chmod system call can be called from Fortran as a function (on all systems except Cray MPP systems and CRAY T90 series systems):

```
CHARACTER*n path
INTEGER CHMOD, mode, I
I = CHMOD (path, mode)
```

Alternatively, chmod can be called from Fortran as a subroutine (on all systems except Cray MPP systems and CRAY T90 series systems). In this case, the return value of the system call is unavailable.

```
CHARACTER*n path
INTEGER mode
CALL CHMOD (path, mode)
```

The Fortran program cannot specify both the subroutine call and the function reference to chmod from the same procedure. *path* may also be an integer variable. In this case, the data must be packed 8 characters per word and terminated with a null (0) byte. The PXFCHMOD(3F) subroutine provides similar functionality and is available on all Cray Research systems.

EXAMPLES

The following examples illustrate different uses of the chmod system call.

Example 1: The chmod system call grants read/write permission to the owner of file1 and only read permission to all other users:

```
if (chmod("file1", 0644) == -1) {
     perror("chmod failed");
}
```

Example 2: Setting a file's setgid bit (02000) and clearing the group execute permission bit enables mandatory file locking for the file. This chmod call establishes mandatory file locking for the datafile file in addition to granting the other file access permissions.

```
if (chmod("datafile", 02644) == -1) {
     perror("chmod setting mandatory locking failed");
}
```

SEE ALSO

chkpnt(2), chown(2), creat(2), fcntl(2), listio(2), mknod(2), open(2), read(2), reada(2), stat(2), trunc(2), write(2), writea(2)

PXFCHMOD(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

chown, lchown, fchown – Changes owner and group of a file

SYNOPSIS

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

int chown (const char *path, uid_t owner, gid_t group);

int lchown (const char *path, uid_t owner, gid_t group);

int fchown (int fildes, uid_t owner, gid_t group);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The chown, lchown, and fchown system calls assign a new owner and group to a file. These system calls accept the following arguments:

path Points to a file path name.

owner Specifies the numeric value of the new owner ID.

group Specifies the numeric value of the new group ID.

fildes Specifies the file descriptor.

If the POSIX_CHOWN_RESTRICTED option is enabled, only a process with appropriate privilege may change file ownership.

If the process does not have appropriate privilege, then the file S_ISUID (set-user-ID) and S_ISGID (set-group-ID) mode bits are cleared.

If *path* is a symbolic link, chown will change the owner and group of the file referenced by the symbolic link. lchown will change the owner and group of the symbolic link itself.

If *owner* or *group* is specified as −1, the corresponding ID of the file is not changed.

Only the owner of a file or a process with appropriate privilege may change file ownership.

Only a process with appropriate privilege can change the owner of a restart file.

NOTES

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

The process must be granted write permission to the file via the security level and compartments. That is, the active security label of the process must equal the security label of the file.

If the FSETID_RESTRICT configuration option is enabled, only a process with appropriate privilege is allowed to change the owner of set-user-ID or set-group-ID files.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_CHOWN	If the POSIX_CHOWN_RESTRICTED option is enabled, the process is allowed to change file ownership.
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_FOWNER	The process is considered the file owner.
PRIV_FSETID	The process is allowed to change the owner of a set-user-ID or set-groups-ID file.
PRIV_FSETID	The process is allowed to preserve the set-user-ID or set-groups-ID mode bits.
PRIV_MAC_READ	The process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_WRITE	The calling process is granted write permission to the file via the security label.
PRIV_RESTART	The process is allowed to set the mode of a restart file.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix. The super user is considered the file owner and is allowed to change the owner of a restart file. The super user is allowed to preserve the set-user-ID and set-groups-ID mode bits. If the POSIX_CHOWN_RESTRICTED option is enabled, the super user or a process with the PERMBITS_CHOWN permbit is allowed to change file ownership.

If the PRIV_SU configuration option is enabled, the super user is granted write permission to the file via the security label. If the PRIV_SU and FSETID_RESTRICTED configuration options are enabled, the super user is allowed to change the owner of a set-user-ID and set-groups-ID file.

RETURN VALUES

When chown, lchown, or fchown completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The chown, lchown, or fchown system call fails and the owner and group of the specified file remains unchanged if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The *path* argument points outside the allocated address space of the process.
EMANDV	User's security label is not equal to that of the file.
EMANDV	The calling process does not have MAC write access to the file to which the file descriptor refers.
EMANDV	The process is not granted write permission to the file via the security label.
ENOENT	The specified file does not exist.
ENOTDIR	A component of the path prefix is not a directory.
EPERM	The process is not a file owner and does not have appropriate privilege.
EPERM	The file is a restart file and the process does not have appropriate privilege.
EPERMIT	User is a trusted user, but does not have suidgid permission.
EPERMIT	If the FSETID_RESTRICT configuration option is enabled, the process does not have appropriate privilege to change the owner of a set-user-ID and set-group-ID file.
EQGRP	A file or inode quota limit was reached for the new group ID.
EQUSR	A file or inode quota limit was reached for the new user ID.
EROFS	The specified file resides on a read-only file system.

FORTRAN EXTENSIONS

The chown system call can be called from Fortran as a function (on all systems except Cray MPP systems and CRAY T90 series systems):

```
CHARACTER*n path
INTEGER CHOWN owner, group, I
I = CHOWN (path, owner, group)
```

Alternatively, chown can be called from Fortran as a subroutine (on all systems except Cray MPP systems and CRAY T90 series systems). In this case, the return value of the system call is unavailable.

```
CHARACTER*n path
INTEGER owner, group
CALL CHOWN (path, owner, group)
```

The Fortran program cannot specify both the subroutine call and the function reference to chown from the same procedure. *path* may also be an integer variable. In this case, the data must be packed 8 characters per word and terminated with a null (0) byte. The PXFCHOWN(3F) subroutine provides similar functionality and is available on all Cray Research systems.

EXAMPLES

This example shows how the chown request changes ownership on a file (chown is a restricted operation for most users).

The getpwnam (see getpwent(3C)) library routine first locates the user and group IDs for user joe from the /etc/passwd file. chown changes the ownership of file myfile to user joe.

```
#include <pwd.h>
#include <unistd.h>

main()
{
     struct passwd *pwptr;

     pwptr = getpwnam("joe");
     if (chown("myfile", pwptr->pw_uid, pwptr->pw_gid) == -1) {
          perror("chown failed");
     }
}
```

FILES

/usr/include/sys/types.h	Contains types required by ANSI X3J11
/usr/include/unistd.h	Contains C prototype for the chown, fchown, and lchown system calls

SEE ALSO

chkpnt(2), chmod(2)

getpwnam(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

PXFCHOWN(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

chroot – Changes the root directory

SYNOPSIS

```
#include <unistd.h>
int chroot (const char *path);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

XPG4

DESCRIPTION

The chroot system call causes the specified directory to become the root directory; that is, the starting point for searches for path names beginning with a slash (/). It accepts the following argument:

path Points to a directory path name. The chroot system call does not affect your working directory.

The process must have appropriate privilege to use this system call. For more information on permission bits (permbits), see *General UNICOS System Administration*, Cray Research publication SG–2301.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, you cannot use .. to access files outside the subtree rooted at the root directory.

NOTES

To be granted search permission to a component of the path, the active security label of the process must be greater than or equal to the security label of the component.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_ADMIN	The process is allowed to use this system call.
PRIV_DAC_OVERRIDE	The process is granted search permission to a component of the path via the permission bits and access control list.
PRIV_MAC_READ	The process is granted search permission to a component of the path via the security label.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_CHROOT permbit is allowed to use this system call. The super user is granted search permission to every component of the path.

RETURN VALUES

When `chroot` completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

ERRORS

The `chroot` system call fails and the root directory remains unchanged if one of the following error conditions occurs:

Error Code	Description
`EACCES`	The process is denied search permission to a component of the specified path.
`EFAULT`	The *path* argument points outside the allocated address space of the process.
`ENOENT`	The specified directory does not exist.
`ENOTDIR`	Any component of the path name is not a directory.
`EPERM`	The process does not have appropriate privilege to use this system call.

FILES

`/usr/include/unistd.h`	Contains C prototype for the `chroot` system call

SEE ALSO

`chdir`(2)

`udbgen`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

General UNICOS System Administration, Cray Research publication SG–2301

NAME

close – Closes a file descriptor

SYNOPSIS

```
#include <unistd.h>
int close (int fildes);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The close system call closes the specified file descriptor. It accepts the following argument:

fildes Specifies a file descriptor. It is obtained from an accept(2), creat(2), dup(2), fcntl(2), open(2), pipe(2), socket(2), or socketpair(2) system call.

RETURN VALUES

If close completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The close system call fails if one of the following error condition occurs:

Error Code	Description
EBADF	The *fildes* argument is not a valid open file descriptor.
EINTR	The close system call was interrupted.

FILES

/usr/include/unistd.h	Contains C prototype for the close system call

SEE ALSO

accept(2), creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2), shutdown(2), socket(2), socketpair(2)

NAME

cmptext – Compares the supplied character sequence with the privilege text of the calling process

SYNOPSIS

```
#include <sys/priv.h>
#include <sys/tfm.h>

int cmptext (long *seq, long flags);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The cmptext system call compares the supplied character sequence with the privilege text of the calling process.

The cmptext system call accepts the following arguments:

seq Specifies the character sequence. The character sequence is a long integer but can be passed as a character sequence enclosed by single quotation marks (for example, cmptext('*charseq*',...).

flags Specifies a bit mask that can contain any combination of the ROOT_EFFECTIVE and ROOT_REAL flags. The flags are defined as follows:

Flag	Description
ROOT_EFFECTIVE	Indicates that any process whose effective user ID is 0 automatically has the supplied privilege text when PRIV_SU is enabled.
ROOT_REAL	Indicates that any process whose real user ID is 0 automatically has the supplied privilege text when PRIV_SU is enabled.

A *flags* value that includes neither ROOT_EFFECTIVE or ROOT_REAL indicates that user ID 0 does not automatically have the specified privilege text. A *seq* value of 0 causes cmptext to return successfully if the calling process has null privilege text.

RETURN VALUES

A return value of 0 indicates that the privilege text of the calling process is identical to the supplied character sequence, or that the caller meets the user ID 0 requirements as specified previously.

A positive return value indicates that the supplied character sequence does not match the privilege text of the process and does not meet the user ID 0 requirements specified previously.

A return value of –1 indicates that an error has occurred, and an error code is stored in `errno`.

ERRORS

The `cmptext` system call fails if the following error condition occurs:

Error Code	Description
`EINVAL`	A value supplied for *seq* or *flags* was not valid.

NAME

connect – Initiates a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (int s, struct sockaddr *name, int namelen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The connect system call initiates a connection on a socket. It accepts the following arguments:

s — Specifies the descriptor of the socket to connect. When the socket is of the SOCK_RAW or SOCK_DGRAM type, the connect system call permanently specifies the peer to which datagrams are sent. If the socket is of the SOCK_STREAM type, this call tries to connect to another socket.

name — Points to a sockaddr structure that contains the destination address of the socket to which the *s* socket is to be connected. This destination address is in the address domain of the socket.

namelen — Specifies the length of the destination address. The length is measured in bytes.

Each address domain uniquely interprets the *name* argument. Generally, stream sockets can successfully connect only once; datagram sockets can use the connect call multiple times to change association. Datagram connections can dissolve an association by connecting to an invalid address such as a null address.

NOTES

If *namelen* is less than the size of the address of the connecting entity (that is, less than the size of a struct sockaddr), the accept(2) system call truncates its result to fit into the available space.

The connect system call is subjected to additional security rules. The two sockets being connected each have security attributes that are inherited from their associated processes. These attributes must be equal if the SOCKET_MAC option is enabled. In addition, the network and remote host have security-attribute ranges, which are specified in the network access list (NAL) portion of the spnet.conf configuration file and administered by using the spnet(8) command.

If the SOCKET_MAC configuration option is not enabled, the security attributes of the socket are not required to be equal, but the security range of the process, which is specified in the UDB for the user, must include the minimum label for the remote host as specified in the NAL. SOCKET_MAC is part of TCP/IP configurable feature variables list in uts/cf/Nmakefile.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_MAC_WRITE	The process is allowed to override the security label restrictions when the SOCKET_MAC option is enabled.

If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions when the SOCKET_MAC option is enabled.

RETURN VALUES

If connect completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and errno is set to indicate the error.

ERRORS

The connect system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	If the SOCKET_MAC configuration option is enabled, the process does not meet the security label requirements and does not have appropriate privilege.
EADDRINUSE	Address is already in use.
EADDRNOTAVAIL	Specified address is unavailable on this machine.
EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
EBADF	Descriptor *s* is invalid.
ECONNREFUSED	Attempt to connect is forcefully rejected.
EFAULT	Argument *name* specifies an area outside the process address space.
EISCONN	Socket is already connected.
ENETUNREACH	Network cannot be reached from this host.
ENOTSOCK	Descriptor *s* is not a socket.
ETIMEDOUT	Connection establishment timed out without establishing a connection.
EWOULDBLOCK	Socket is nonblocking; the connection cannot be completed immediately. You can use the select(2) call to select the socket while it is connecting by selecting it for writing.

EXAMPLES

This client program shows how to use the connect system call in context with other TCP/IP calls. The program creates a TCP/IP socket and then attempts to establish a connection between the newly created socket and the socket within the server program on the designated server host. If a connection is successful, the client process sends a string of data to the server process.

```
/*  Client side of client/server socket example. For server side,
    see socket(2).
    Syntax: client hostname portnumber */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

/*      in in.h is this socket structure
 *
 *       Socket address, internet style.
 *
 *       struct sockaddr_in {
 *          short   sin_family;
 *          u_short sin_port;
 *          struct  in_addr sin_addr;
 *          char    sin_zero[8];
 *      };
 */

#define DATA "Test message from client to server."

main(int argc, char *argv[])
{
     int s;
     struct sockaddr_in dest;          /* destination socket address */
     struct hostent *hp;               /* host structure pointer */

/* Converts host name into network address. */
     hp = gethostbyname(argv[1]);

     dest.sin_family = hp->h_addrtype; /* addr type (AF_INET) */
     bcopy(hp->h_addr_list[0], &dest.sin_addr, hp->h_length);
     dest.sin_port = atoi(argv[2]);

     /* create port */

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("client, cannot open socket");
          exit(1);
     }
     if (connect (s, (struct sockaddr *) &dest, sizeof(dest)) < 0) {
```

```
                close(s);
                perror("client, connect failed");
                exit(1);
           }
           write(s, DATA, sizeof(DATA));

           close(s);
           exit(0);
      }
```

FILES

`/etc/config/spnet.conf`	Network access list file
`/usr/adm/sl/slogfile`	Receives security log records
`/usr/include/sys/socket.h`	Contains definitions related to sockets, types, address families, and options
`/usr/include/sys/types.h`	Contains types required by ANSI X3J11

SEE ALSO

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

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

`cpselect` – Selects which processors may run the process

SYNOPSIS

```
#include <unistd.h>
int cpselect (int pid, int mask);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The `cpselect` system call specifies the physical processors that may execute a process as specified by the following arguments: It accepts the following arguments:

pid Specifies the process ID of process to execute. A *pid* of 0 means the current process.

mask Specifies the bit mask indicating physical processors. Processor A or 0 is 01, processor B or 1 is 02, and so on. A *mask* of 0 signifies the use of any available CPU; a *mask* of −1 does not change the select mask, but it returns the previous mask. The CPU mask is inherited by child processes.

NOTES

The *mask* argument is silently limited to the available processors. If all processors chosen are unavailable, the process is allowed to run on any processor.

The active security label of the calling process must be greater than or equal to the security label of the affected process.

To set the processor execution mask of a process, the active security label of the calling process must equal the security label of the affected process.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
`PRIV_MAC_READ`	The active security label of the calling process is considered greater than or equal to the security label of the affected process.
`PRIV_MAC_WRITE`	The active security label of the calling process is considered equal to the security label of the affected process.
`PRIV_POWNER`	The calling process is considered the owner of the affected process.

If the `PRIV_SU` configuration option is enabled, the super user is considered to be the owner of the affected process. If the `PRIV_SU` configuration option is enabled, the super user is allowed to bypass security label restrictions.

RETURN VALUES

If `cpselect` completes successfully, it returns the previous *mask* value; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

ERRORS

The `cpselect` system call fails if one of the following error conditions occurs:

Error Code	Description
`EINVAL`	The *pid* argument contains an invalid value.
`EPERM`	The real or effective user ID of the calling process does not match the real or effective user ID of the affected process, and the calling process does not have appropriate privilege.
`ESRCH`	No process can be found to match the *pid*.

FORTRAN EXTENSIONS

The `cpselect` system call can be called from Fortran as a function:

```
INTEGER pid, mask, CPSELECT, I
I = CPSELECT (pid, mask)
```

Alternatively, `cpselect` can be called from Fortran as a subroutine. In this case, the return value of the system call is unavailable.

```
INTEGER pid, mask
CALL CPSELECT (pid, mask)
```

The Fortran program cannot specify both the subroutine call and the function reference to `cpselect` from the same procedure.

FILES

`/usr/include/unistd.h`	Contains C prototype for the `cpselect` system call

NAME

creat – Creates a new file or rewrites an existing one

SYNOPSIS

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

int creat (const char *path, mode_t mode);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The creat system call creates a new ordinary file or prepares to rewrite an existing file. The call is equivalent to an open(2) system call of the following form:

```
open (path, O_WRONLY|O_CREAT|O_TRUNC, mode)
```

For a description of the arguments used in this call, see the open(2) man page.

The creat system call accepts the following arguments:

path Points to a file to be created or an existing file to be rewritten.

mode Specifies the bit pattern denoting the file's access permission. (See stat(2) for the description of these bits.)

If the file exists, the length is truncated to 0 and the mode and owner are unchanged. Otherwise, the file's owner ID is set to the effective user ID of the process, and the group ID is set to the group ID of the directory in which the file is created. The low-order 12 bits of the file mode are set to the value of *mode* modified as follows: all bits set in the process' file mode creation mask are cleared. See umask(2).

If creat completes successfully, the file descriptor is returned, and the file is opened for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across exec(2) system calls (see fcntl(2)). No process can have more than OPEN_MAX files open simultaneously. You can create a new file with a mode that forbids writing.

NOTES

The active security label of the calling process must fall within the security label range of the file system on which the new file will reside.

If the FSETID_RESTRICT option is enabled, only a process with appropriate privilege can create set-user-ID or set-group-ID files.

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

To be granted write permission to the parent directory, the active security label of the process must equal the security label of the directory.

To be granted write permission to an existing file, the active security label of the process must equal the security label of the file.

A process with the effective privilege shown is granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_DAC_OVERRIDE	The process is granted write permission to the parent directory or to an existing file via the permission bits and access control list.
PRIV_FSETID	When the FSETID_RESTRICT option is enabled, the process is allowed to create set-user-ID or set-group-ID files.
PRIV_MAC_READ	The process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_WRITE	The process is granted write permission to the parent directory or to an existing file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted write permission to the parent directory and to an existing file. When FSETID_RESTRICT is enabled, the super user is allowed to create set-user-ID and set-group-ID files.

RETURN VALUES

If creat completes successfully, a nonnegative integer, (the file descriptor) is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The creat system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	The file exists and write permission is denied.

EACCES	The file does not exist, and the directory in which the file is to be created does not permit writing.
EACCES	Search permission is denied on a component of the path prefix.
EAGAIN	The file exists, mandatory file and record locking is set, and outstanding record locks exist on the file (see chmod(2)).
EFAULT	The *path* argument points outside the allocated address space of the process.
EFLNEQ	The active security label of the calling process does not fall within the range of the file system on which the new or rewritten file will reside.
EISDIR	The specified file is an existing directory.
EMFILE	OPEN_MAX file descriptors are currently open.
ENFILE	The system file table is full.
ENOENT	A component of the path prefix does not exist.
ENOENT	The path name is null.
ENOTDIR	A component of the path prefix is not a directory.
EQACT	A file or inode quota limit was reached for the current account ID.
EQGRP	A file or inode quota limit was reached for the current group ID.
EQUSR	A file or inode quota limit was reached for the current user ID.
EROFS	The specified file resides or would reside on a read-only file system.
ETXTBSY	The text file is busy.
EWRITV	If the FSETID_RESTRICT option is enabled, the process does not have appropriate privilege to create a set-user-ID or set-group-ID file.

FORTRAN EXTENSIONS

The creat system call can be called from Fortran as a function (on all systems except Cray MPP systems and CRAY T90 series systems):

```
CHARACTER*n path
INTEGER  mode, CREAT, I
I =  CREAT (path, mode)
```

Alternatively, creat can be called from Fortran as a subroutine (on all systems except Cray MPP systems and CRAY T90 series systems). In this case, the return value of the system call is unavailable.

```
CHARACTER*n path
INTEGER mode
CALL CREAT (path, mode)
```

The Fortran program must not specify both the subroutine call and the function reference to `creat` from the same procedure. *path* may also be an integer variable. In this case, the data must be packed 8 characters per word and terminated with a null (0) byte. The PXFCREAT(3F) subroutine provides similar functionality and is available on all Cray Research systems.

SEE ALSO

`chmod`(2), `close`(2), `dup`(2), `exec`(2), `fcntl`(2), `lseek`(2), `open`(2), `read`(2), `stat`(2), `umask`(2), `write`(2)

PXFCREAT(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

cutimes – Updates user execution time

SYNOPSIS

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

struct utms *cutimes (struct utms *mytimes);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The cutimes system call lets users have a structure in user memory continually updated with user execution time information. From this utms structure, users can determine how much execution time has accumulated in an interval without calling the operating system.

The cutimes system call accepts the following argument:

mytimes Specifies the address of the structure to receive the data.

If the address is 0, the structure is no longer updated.

The utms structure contains the following members:

```
time_t   utms_update;  /* RT clock at start of this connect */
time_t   utms_utime;   /* Total user time during previous connects */
```

Since the utms structure is only updated at the time of connection to a process, the current user time accumulated since the process began can be calculated as follows:

```
time = mytimes.utms_utime + (rtclock() - mytimes.utms_update);
```

This method will yield the desired results most of the time. But there is a small chance that the operating system will change the utms values in the middle of the calculation. A guaranteed method is shown in the EXAMPLES section and is also implemented in the cpused(3C) function.

Update of the utms structure stops if one of the following occurs:

- A memory contraction places the structure outside user memory.
- An exec(2) system call is executed.

NOTES

The times in the utms structure are figured from the time the process began execution, not from the invocation of the cutimes call. Monitoring the utms structure at the user level is somewhat tricky because the user may be interrupted when looking at the structure. The cpused(3C) routine monitors this structure.

RETURN VALUES

If `cutimes` completes successfully, the address of the user structure is returned, with 0 meaning that the feature is disabled. Otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `cutimes` system call fails if the following error condition occurs:

Error Code	Description
`EFAULT`	The *mytimes* argument is out of the user's memory space.

EXAMPLES

This example shows how to use the `cutimes` system call to compute the amount of user execution time for a section of code within a user's program. The amount of user time is computed in hardware clock ticks and seconds.

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

main()
{
     struct utms mytimes, sample;
     time_t utime, rt, before, after;

     cutimes(&mytimes);                          /* enable execution time
                                                    update feature */

     do {
          rt = rtclock();
          sample = mytimes;                      /* sample mytimes before */
     } while (rt < sample.utms_update);
     before = sample.utms_utime + (rt - sample.utms_update);

     /* Section of code here is where user execution time is to be measured. */

     do {
          rt = rtclock();
          sample = mytimes;                      /* sample mytimes after */
     } while (rt < sample.utms_update);
     after = sample.utms_utime + (rt - sample.utms_update);

     utime = after - before;                     /* compute user time -
                                                    measured in clock ticks */
```

```
      printf("\nCPU time used in user space = %f sec or %ld clock ticks\n",
               (float)utime/(float)CLK_TCK, utime);

      cutimes((struct utms *) 0);                  /* disable execution time
                                                      update feature */
}
```

SEE ALSO

exec(2)

cpused(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

second(3F) in the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165

NAME

dacct – Enables or disables process and daemon accounting

SYNOPSIS

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

int dacct (char *path, int did);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The dacct system call enables or disables process and daemon accounting.

When process accounting is enabled, an accounting record is written to an accounting file for each process that terminates. Process termination can be caused by an exit(2) call, a chkpnt(2) call, or receipt of a fatal signal. When a job terminates, an end-of-job record is written.

Similarly, when daemon accounting is enabled, the daemons may write accounting records.

Accounting is disabled when *path* is a null pointer and no errors occur during the system call.

The dacct system call accepts the following arguments:

path Points to the path name of the accounting file, which is defined by acct(5). The daemon accounting files are defined in /usr/include/acct/dacct.h.

did Identifies the type of accounting that is to be enabled or disabled. These daemon identifiers are specified in /usr/include/sys/accthdr.h.

If the specified type of accounting is already enabled and *path* differs from the accounting file currently in use, the accounting file will be switched to *path* without the loss of any accounting information.

Only a process with appropriate privilege can use this system call.

NOTES

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

To be granted write permission to the file, the active security label of the process must equal the security label of the file.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_ACCT	The process is allowed to use this system call.

PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_DAC_OVERRIDE	The process is granted write permission to the file via the permission bits and access control list.
PRIV_MAC_READ	The calling process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_WRITE	The process is granted write permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted write permission to the file. The super user is allowed to use this system call.

RETURN VALUES

If dacct completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The dacct system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied on a component of the path prefix.
EACCES	The process is not granted write permission to the file.
EACCES	The file specified by *path* is not an ordinary file.
EFAULT	The *path* argument points to an illegal address.
EINVAL	An argument that is not valid was passed to the system call.
EISDIR	The *path* argument is a directory.
EPERM	The process does not have appropriate privilege to use this system call.
EROFS	The specified file resides on a read-only file system.

FILES

/usr/include/acct/dacct.h	Defines daemon accounting files
/usr/include/sys/accthdr.h	Specifies daemon identifiers

SEE ALSO

acct(2), chkpnt(2), exit(2)

acct(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

devacct – Controls device accounting

SYNOPSIS

```
#include <sys/types.h>
#include <sys/acct.h>
int devacct (char *device, int func, int type);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Only a process with appropriate privilege can use this system call. The devacct system call accepts the following arguments:

device Specifies the name of a block special device that is a local file system. This name is used only when the ACCT_LABEL function is specified.

func This argument can be one of the following:

- ACCT_ON — Turns on accounting for requested type.
- ACCT_OFF — Turns off accounting for requested type.
- ACCT_LABEL — Labels the *device* with the label indicated by *type*. You can label only block special devices.

type Specifies device type. For block special devices, valid values are 0 to (MAXBDEVNO – 1). For character special devices, the values are 0 to (MAXCDEVNO – 1) OR'ed with ACCT_CHSP. For performance accounting, *type* is ACCT_PERF OR'ed with PERF_01.

See the /etc/config/acct_config file for the block and character device types. ACCT_PERF and PERF_01 are defined in /usr/include/sys/acct.h.

NOTES

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

The process must be granted write permission to the device file via the security label. That is, the active security label of the process must equal the security label of the device file.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_ACCT	The process is allowed to use this system call.

PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_MAC_READ	The calling process is granted search permission to every component of the path prefix via the security label.
PRIV_WRITE	The process is granted write permission to the device file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted write permission to the file. The super user is allowed to use this system call.

If the PRIV_SU configuration option is enabled, the super user is granted write permission to the device file via the security label.

RETURN VALUES

The devacct system call returns the previous accounting type when called to label a device. It returns the previous state, ACCT_ON or ACCT_OFF, when called to turn device accounting on or off. Otherwise, a value of −1 is returned, and errno is set to indicate the error.

ERRORS

The devacct system call fails if one of the following error conditions occurs:

Error Code	Description
EINVAL	The *type* on the label request is bad.
EINVAL	The device is not a block special on a label request.
EINVAL	The device label is not legal for an on or off request.
EINVAL	The *func* argument is not ACCT_ON, ACCT_OFF, or ACCT_LABEL.
ENODEV	The device is not mounted on a label request.
EPERM	The process does not have appropriate privilege to use this system call.
EPERM	The process is not granted write permission to the file via the security label.

EXAMPLES

The following examples illustrate different uses of the devacct system call. (You also can obtain the functionality in these examples by using the devacct(8) command.)

Example 1: This example shows how to label the block device /dev/dsk/root as a DD−40 device. The numeric type for a DD−40 is 2 because a DD−40 disk drive is associated with the BLOCK_DEVICE2 variable in /etc/config/acct_config.

```
devacct("/dev/dsk/root",ACCT_LABEL,2);
```

Example 2: This example shows how to turn on DD−40 device accounting:

```
devacct(0,ACCT_ON,2);
```

Example 3: This example shows how to turn on performance accounting:

```
devacct(0,ACCT_ON,ACCT_PERF|PERF_01);
```

FILES

`/etc/config/acct_config`	Accounting configuration file
`/usr/include/sys/acct.h`	Defines daemon accounting files
`/usr/include/sys/param.h`	Defines configuration files

SEE ALSO

`devacct`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR−2022

NAME

dmmode – Sets and gets data migration retrieval mode

SYNOPSIS

```
#include <unistd.h>
int dmmode (int mode);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The dmmode system call sets the data migration retrieval mode of the calling process. It accepts the following argument:

mode Specifies the mode. dmmode set the data migration retrieval mode of the calling process and returns the previous value of the mode. Only the low-order 9 bits of *mode* are used.

A nonzero data migration retrieval mode specifies that offline files are retrieved automatically as soon as they are accessed (see open(2)). A value of 0 specifies that files must be explicitly recalled (see dmget(1)) before they can be accessed successfully. In this mode, an access attempt on a migrated file results in the return of an error code.

RETURN VALUES

The previous value of the data migration retrieval mode is returned.

FILES

/usr/include/unistd.h	Contains C prototype for the dmmode system call

SEE ALSO

open(2)

dmget(1), dmlim(1), dmput(1) Online only

dmmctl(8) Online only

NAME

dmofrq – Processes offline file requests

SYNOPSIS

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

int dmofrq (void *fptr, int cmd, void *arg, struct dmo_hand *hand,
long *rcode);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The data migration daemon uses the dmofrq system call to process requests related to offline files. Only an appropriately privileged process is allowed to use this system call. You should not use dmofrq in your programs; dmofrq is intended for use only by data migration packages, such as the Cray Data Migration Facility (DMF).

The dmofrq system call accepts the following arguments:

fptr Selects the offline file. The *fptr* argument is either a character pointer to the path name or a word pointer to a dm_dvino structure with the device and inode number of the offline file specified.

cmd Selects the type of request to be processed. *cmd* is specified as a character: a lowercase character indicates that *fptr* is a path; an uppercase character indicates that *fptr* is a pointer to a dm_dvino structure.

arg Points to an argument required by the command.

hand Specifies the handle for a file to be migrated or recalled.

rcode Specifies a word into which a return code is written by dmofrq. The valid return codes and their meanings are defined in dmofrq.h.

The *hand* and *rcode* arguments are used only by the migrate and recall commands (a, A, b, B, e, E, f, F, g, G, u, U, v, and V).

The valid forms of the *cmd* and *arg* arguments are as follows:

a or A Begins remigration of a dual-state file.

b or B Begins migration of a file.

c or C Changes the migration status of a file. The arg parameter is a pointer to a dm_file_change structure. When the call is processed, if the file matches the before and gen fields, the migration attributes are changed to the values in the after structure. *hand* and *rcode* should be NULL.

d or D Simulates the secstat(2) system call. *arg* is a pointer to a secstat stucture, and *hand* and *rcode* should be NULL.

e or E — Begins remigration of a dual-state file, except that *arg* is a pointer to a `dm_dvino` structure.

f or F — Begins migration of a file, except that the *arg* parameter is a pointer to a `dm_dvino` structure.

g or G — Completes migration of a file. The file is changed to offline and the data blocks are swapped to the destination file and released. *arg* is a pointer to a `dm_dvino` structure.

l or L — Writes the file size (to which *arg* points) to the offline file inode. *arg* is a pointer to a `long`, and *hand* and *rcode* should be `NULL`.

s or S — Simulates the `stat`(2) system call for the indicated file. The status information is returned to the area to which *arg* points. (The lowercase `s` command is obsolete; it will be removed in a future release of UNICOS.) *arg* is a pointer to a `stat` structure, and *hand* and *rcode* should be `NULL`.

w or W — Writes the file handle (to which *arg* points) to the offline file inode. *arg* is a pointer to a `dmo_hand` structure, and *hand* and *rcode* should be `NULL`.

u or U — Completes file recall and zeroes the inode's handle. *arg* is the path of the file containing the disk blocks.

v or V — Completes file recall without zeroing the inode's handle. The handle of the *fptr* file is not cleared; instead, *fptr* becomes a dual-state file with a valid migration handle. *arg* is the path of the file containing the disk blocks.

The `dmofrq` system call receives information from the following structures:

```
typedef struct  dm_dvino {
        dev_t   dm_dev;         /* device number */
        ino_t   dm_ino;         /* inode number */
} dm_dvino_t;

typedef struct  dmo_hand {
        uint    dmport:3;       /* daemon number */
        uint    dmstate:5;      /* file status */
        uint    dmunused1:24;   /* unused */
        uint    machid:32;      /* offline file machine id */
        long    ofilenm;        /* offline file number */
} dmo_hand_t;

typedef struct file_dm_state {
      long    size;             /* File size in bytes */
      long    dm_mid;           /* offline file machine id */
      long    dm_key;           /* offline file number */
      long    dm_state;         /* migration state */
      int     dm_port;          /* daemon number */
} file_dm_state_t;

typedef struct file_dm_change {
      long            gen;    /* File generation number */
```

```
        file_dm_state_t before; /* current state of the file */
        file_dm_state_t after;  /* State of file after the change */
} file_dm_change_t;
```

The dm_dvino structure contains device and inode information, and the dmo_hand structure contains handle, state, and port information.

NOTES

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_ADMIN	The process is allowed to use this system call.
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the file path prefix via the permission bits and access control list.
PRIV_MAC_READ	The process is granted search permission to every component of the file path prefix via the security label.
PRIV_MAC_READ	The process is allowed to override the file security label protections when performing a stat(2) or secstat(2) operation.

If the PRIV_SU configuration option is enabled, the super user is allowed to perform all dmofrq operations on any file.

CAUTIONS

You should not use this system call in your programs. It is intended to be used by data migration packages such as DMF only.

RETURN VALUES

If dmofrq completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error. If errno is set to EDMOFRQ, *rcode* will contain a detailed error return code. The valid *rcode* return values and their meanings are defined in the sys/dmofrq.h file.

ERRORS

The dmofrq system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	Search permission is denied for a component of the path prefix.
EBUSY	The file to be changed is in use.
EDMOFF	The data migration system is not configured.
EDMOFRQ	Inconsistent or invalid parameters or file attributes exist. See the value returned to *rcode* for a detailed specification.

EDMRBPAR	An invalid *cmd* was specified.
EDMRNOLF	The file indicated by *fptr* was not an offline S_IFOFL or a regular S_IFREG file.
EDMRNSD	The *fptr* and *cmd* files were on different file systems.
EDMRWFT	The file owning the data blocks was not a regular file.
EFAULT	A pointer is not valid.
EINVAL	The device and inode do not point to a valid inode.
ENOENT	The specified file does not exist.
ENOTDIR	A component of the path prefix is not a directory.
EPERM	The process does not have appropriate privilege to use this system call.

FILES

/usr/include/sys/dmofrq.h	Contains definitions related to data migration

SEE ALSO

dmmode(2) for information about setting and getting data migration retrieval mode
secstat(2) for information about getting file security attributes
stat(2) for information about getting file status

NAME

dup – Duplicates an open file descriptor

SYNOPSIS

```
#include <unistd.h>
int dup (int fildes);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The dup system call duplicates an open file descriptor. It accepts the following argument:

fildes Specifies a file descriptor. It is obtained from a creat(2), dup, fcntl(2), open(2), or pipe(2) system call.

The dup system call returns a new file descriptor having the following characteristics in common with the original:

- Same open file (or pipe)
- Same file pointer (that is, both file descriptors share one file pointer)
- Same access mode (read, write, or read/write)

The new file descriptor is set to remain open across exec(2) system calls. See fcntl(2).

The file descriptor returned is the lowest one available.

RETURN VALUES

If dup completes successfully, a nonnegative integer (the file descriptor) is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The dup system call fails if one of the following error conditions occurs:

Error Code	Description
EBADF	The *fildes* argument is not a valid open file descriptor.
EMFILE	OPEN_MAX file descriptors are currently open.

FORTRAN EXTENSIONS

The dup system call can be called from Fortran as a function:

```
INTEGER fildes, DUP, I
I = DUP (fildes)
```

Alternatively, dup can be called from Fortran as a subroutine. In this case, the return value of the system call is unavailable.

```
INTEGER fildes
CALL DUP (fildes)
```

The Fortran program must not specify both the subroutine call and the function reference to dup from the same procedure.

EXAMPLES

The following illustrates how the dup system call is used by a shell program to provide for the UNICOS user-level redirection (>) feature.

The user enters the following command to the shell program:

```
$ command > newfile
```

To redirect the output of the command to the named file (newfile), the shell closes stdout (file descriptor = 1) and then uses dup to duplicate the file descriptor for newfile. The duplicate file descriptor reuses the file descriptor previously used by stdout. The command then executes writing its data to the file having file descriptor 1, which is the file newfile rather than the typical stdout device.

```
int fd, perms_mask;
char *fptr;

fd = creat(fptr, perms_mask); /* fptr points to requested
                                 file name */
fclose(stdout);
dup(fd);                     /* duplicate file descriptor
                                 replaces stdout */
close(fd);
/* stdout has now been redirected. */
```

FILES

/usr/include/unistd.h	Contains C prototype for the dup system call

SEE ALSO

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

NAME

`exctl` – Exchanges control

SYNOPSIS

```
#include <sys/vm.h>
int exctl (struct vmctxt *pctxt);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The `exctl` system call allows a master process to exchange control to a subordinate task running in a virtual machine environment. The subordinate task may be a test kernel, or it may be a user process running under the control of the test kernel. The subordinate task must be fully contained within the address space of the master process. The `exctl` system call accepts the following argument:

pctxt Points to the `vmctxt` structure.

A `vmctxt` structure (virtual machine context) includes the following members:

```
gxp_t     vm_xp;                        /* Exchange package */
int       vm_saveb[MAXBREGS];           /* B registers */
word      vm_savet[MAXTREGS];           /* T registers */
word      vm_savevm;                    /* Vector mask register */
word      vm_savev[MAXVREGS][MAXVLEN];  /* Vector registers */
word      vm_savevm1;                   /* Vector mask register 1 */
word      vm_vsaved;
label_t   vm_save[3];                   /* for switching */
word      ret_status;                   /* Subordinate task's return status */
word      vm_vmsav;                     /* pw_vmsav word (from PWS structure) */
```

The BA and LA values in the exchange package are modified by the system; therefore, these values must be reset each time `exctl` is called. The BA and LA registers are relative to word 0 of the master process. The value of the P register is relative to BA. When this system call is executed, the real kernel converts the given BA and LA values into absolute addresses, loads the hardware registers with the values from the `vmctxt` structure, and starts execution at the given P address. The `exctl` system call returns to the master process when the subordinate task exits for any of the following reasons:

- Normal exchange interrupt
- Error exchange interrupt
- Program range error interrupt
- Operand range error interrupt
- Floating-point error interrupt

- Programmable clock interrupt
- Register parity error

The master process should examine the saved status and the exchange package flags to determine why the subordinate task exited. On return from the `exctl` system call, the `vmctxt` structure contains the contents of the hardware registers as they were when the subordinate task exited.

NOTES

The Programmable Clock Interrupt has been mapped to the signal `SIGALRM`, sent by the `alarm` system call. The intent is to allow the master process to use the `alarm` function to simulate a real-time clock.

RETURN VALUES

If `exctl` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `exctl` system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	The monitor mode bit (XPM_MM) is set in the exchange package of the `vmctxt` structure of the subordinate process.
EFAULT	The context structure to which *pctxt* points is not fully contained in the master process address space.
EFAULT	The BA, LA, and P registers in the exchange package are not within the process address space.

SEE ALSO

`alarm`(2)

NAME

execl, execv, execle, execve, execlp, execvp – Executes a file

SYNOPSIS

```
#include <unistd.h>

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

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

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

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

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

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

IMPLEMENTATION

Cray PVP systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The exec system call in all its forms transforms the calling process into a new process, which is constructed from an ordinary, executable file called the *new process file*. This file consists of a header and the program images (see a.out(5)). There is no return from a successful exec because the calling process is overlaid by the new process.

An interpreter file begins with a line of the form:

#! *pathname* [*arg*]

The *pathname* argument is the path of the interpreter, and *arg* is an optional argument. When an interpreter file is executed, the system execs the specified interpreter. The *pathname* specified in the interpreter file is passed on as *arg0* to the interpreter. If *arg* is specified in the interpreter file, it is passed as *arg1* to the interpreter. Any setuid or setgid permissions bits that are set for the interpreter file are ignored. The remaining arguments to the interpreter are *arg0* through *argn* of the file that was originally executed.

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

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

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

The arguments are as follows:

path Points to a path name that identifies the new process file.

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

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

envp Specifies an array of character pointers to null-terminated strings, which constitute the environment for the new process. The *envp* argument is terminated by a null pointer. For `execl` and `execv`, the C run-time start-up routine (`$start`) places a pointer to the calling process's environment in the global cell, as follows:

```
extern char **environ;
```

It passes the calling process's environment to the new process.

file Points to the new process file. The path prefix for this file is obtained by a search of the directories passed as the *environment* line:

```
PATH =
```

The environment is supplied by the shell (see `ksh`(1)).

File descriptors open in the calling process remain open in the new process, except for those whose Close-on-exec flag are set; see `fcntl`(2). For file descriptors that remain open, the file pointer is unchanged.

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

For signals set by `sigset`(2), `exec` ensures that the new process has the same signal action for each signal type whose action is `SIG_DFL`, `SIG_IGN`, or `SIG_HOLD` as the calling process. However, if the action is to catch the signal, the action will be reset to `SIG_DFL`, and any pending signal for this type will be held.

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

The shared memory segments attached to the calling process will be released and will not be attached to the new process (see `shmat`(2)).

Any additional processes created by `_tfork`(2) or `tfork`(3C) calls are killed off and the resulting program begins execution as a single process. One of the previous multitasked processes may linger on the kernel until the new process exits to maintain the parent-child relationships. This bug condition will be fixed in the next release.

Profiling is disabled for the new process; see `profil`(2). The new process also inherits the following attributes from the calling process:

- Accounting information from `times`(2): `utime`, `stime`, `cutime`, and `cstime`
- Current working directory
- File mode creation mask (see `umask`(2))
- File size limit (see `ulimit`(2))
- Nice value (see `nice`(2))
- Job ID
- Parent process ID
- Process group ID
- Process ID
- Root directory
- `semadj` values (see `semop`(2))
- Time left until an alarm clock signal (see `alarm`(2))
- Trace flag (see `ptrace`(2) request 0)
- tty group ID (see `exit`(2) and `signal`(2))
- Record locks (see `fcntl`(2) and `lockf`(3C))
- Security values: security label, security label range, active and authorized categories.

NOTES

The process must be granted search permission to every component of the path prefix via the permission bits and access control list. The process must be granted search permission to every component of the path prefix via the security label.

The process must be granted execute permission to the file via the permission bits and access control list. The process must be granted execute permission to the file via the security label.

A process with the effective privileges shown are granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to every component of the path prefix via the permission bits and access control list.
PRIV_DAC_OVERRIDE	The process is granted execute permission to the file via the permission bits and access control list.
PRIV_MAC_READ	The process is granted search permission to every component of the path prefix via the security label.
PRIV_MAC_READ	The process is granted execute permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted execute permission to the file.

RETURN VALUES

If exec returns to the calling process, an error has occurred; a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The exec system call fails and returns to the calling process if one of the following error conditions occurs:

Error Code	Description
EACCES	The new process file is not a regular file.
EACCES	The new process file mode denies execution permission.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix.
EDMOFF	The file is offline, and the data migration facility is not configured in the system.
EFAULT	The new process file is not as long as indicated by the size values in its header.
EFAULT	The *path*, *argv*, or *envp* argument points to an illegal address.
ENODEV	(CRAY T90 systems only) The new process has requested or requires a CPU type that is not currently available on the system. For example, an exec of a binary built on a CRAY C90 system would result in this error if the CRAY T90 system contained only IEEE CPUs.
ENOENT	One or more components of the new process file's path name do not exist.
ENOEXEC	The exec is not an execlp or execvp, and the new process file has the appropriate access permission but an invalid magic number in its header.

ENOMEM	The new process requires more memory than is allowed by the system-imposed maximum MAXMEM.
ENOTDIR	A component of the new process file's path prefix is not a directory.
EOFFLIN	The file is offline, and automatic file retrieval is disabled.
EOFLNDD	The file is offline, and the data management daemon is not currently executing.
EOFLNNR	The file is offline, and it is currently unretrievable.
E2BIG	The number of bytes in the argument list of the new process is greater than the system-imposed limit of ARG_MAX bytes.

EXAMPLES

The following examples show differences in the use of the various forms of exec: execl, execv, execle, execve, execlp, and execvp. In each example, the call overlays the currently executing program with a new program having full path name /tmp/newprog. The new program supplies two arguments having values arg1 and arg2.

Example 1: The execl system call requires that the new program be specified by a full or relative path name. The argument list passed to the new program is specified as a list of strings in the execl request.

The environment existing in the current program is preserved in the new program.

```
execl("/tmp/newprog", "newprog", "arg1", "arg2", 0);
```

Example 2: The execv system call requires that the new program be specified by a full or relative path name. The argument list passed to the new program is specified as an array (vector) of pointers to strings in the execv request.

The environment existing in the current program is preserved in the new program.

```
static char *arguments[] = {"newprog", "arg1", "arg2", 0};

execv("/tmp/newprog", arguments);
```

Example 3: The execle request requires that the new program be specified by a full or relative path name. The argument list passed to the new program is specified as a list of strings in the execle request.

The environment existing in the current program is replaced by a new environment in the new program. This environment is specified as an array (vector) of pointers to strings, where each string consists of an environment variable equated to its value.

In this example, the new environment contains only two variables, ENV1 and ENV2. The execle request completely replaces the existing environment in the current program with the new environment. Therefore, if the current environment is to preserved with some additional environment variables added, a composite environment must be formed.

```
static char *newenv[] = {"ENV1=string1", "ENV2=string2", 0};

execle("/tmp/newprog", "newprog", "arg1", "arg2", 0, newenv);
```

Example 4: The execve system call requires that the new program be specified by a full or relative path name. The argument list passed to the new program is specified as an array (vector) of pointers to strings in the execve request.

The environment existing in the current program is replaced by a new environment in the new program. This environment is specified as an array (vector) of pointers to strings, where each string consists of an environment variable equated to its value.

In this example, the new environment contains only two variables, ENV1 and ENV2. The execve request completely replaces the existing environment in the current program with the new environment. Therefore, if the current environment is to preserved with some additional environment variables added, a composite environment must be formed.

```
static char *arguments[] = {"newprog", "arg1", "arg2", 0};
static char *newenv[] = {"ENV1=string1", "ENV2=string2", 0};

execve("/tmp/newprog", arguments, newenv);
```

Example 5: The execlp request requires that the new program be specified by a file name rather than a full or relative path name (as used in the previous examples). UNICOS looks for the specified file by searching the list of directories included in the user's PATH environment variable. As a result of this search, UNICOS could find a program, which is not the intended one, having the requested name. This means the user assumes a greater degree of risk when using the execlp request; it is especially dangerous if the calling program is a setuid (set-user-ID) program.

The argument list passed to the new program is specified as a list of strings in the execlp request.

The environment existing in the current program is preserved in the new program.

```
execlp("newprog", "newprog", "arg1", "arg2", 0);
```

Example 6: The execvp system call requires that the new program be specified by a file name rather than a full or relative path name (as used in previous exec examples). UNICOS looks for the specified file by searching the list of directories included in the user's PATH environment variable. UNICOS could find a program, which is not the intended one, having the requested name. This means the user must assume a greater degree of risk when using the execvp request; it is especially dangerous if the calling program is a setuid (set-user-ID) program.

The argument list passed to the new program is specified as an array (vector) of pointers to strings in the execvp request.

The environment existing in the current program is preserved in the new program.

```
static char *arguments[] = {"newprog", "arg1", "arg2", 0};

execvp("newprog", arguments);
```

FILES

`/usr/include/unistd.h`	Contains C prototype for the `exec` system call

SEE ALSO

`alarm`(2), `chmod`(2), `exit`(2), `fcntl`(2), `fork`(2), `nice`(2), `profil`(2), `ptrace`(2), `semop`(2), `shmat`(2), `signal`(2), `sigset`(2), `_tfork`(2), `times`(2), `ulimit`(2), `umask`(2)

`ksh`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`lockf`(3C), `tfork`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`a.out`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

exit, _exit, newexit, _newexit, _lwp_exit, globalexit, _globalexit, localexit, _localexit, _threadexit – Terminates process

SYNOPSIS

All Cray Research systems:

```
#include <stdlib.h>
void exit (int status);
#include <unistd.h>
void _exit (int status);
```

Cray PVP systems:

```
#include <stdlib.h>
void newexit (int status);
#include <unistd.h>
void _newexit (int status);
void _lwp_exit (int status);
```

Cray MPP systems:

```
#include <stdlib.h>
void globalexit (int status);
void localexit (int status);
#include <unistd.h>
void _globalexit (int status);
void _localexit (int status);
void _threadexit (int status);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4 (applies only to exit)

DESCRIPTION

The `exit` system call terminates the calling process. It accepts the following argument:

status Specifies the exit status of the process. It is returned to the process's parent process.

The process termination has the following consequences:

- All of the file descriptors open in the calling process are closed.
- If the parent process of the calling process is executing a `wait`(2) system call, it is notified that the calling process has terminated and the low-order 8 bits (that is, bits 0377) of *status* are made available to it; see `wait`(2).
- If the parent process of the calling process is not executing a `wait`(2) system call, the calling process is transformed into a zombie process, which is a process that occupies only a slot in the process table; it has no other space allocated in either the user or the kernel space.
- The parent process ID of all existing child processes and zombie processes of the calling process is set to 1. This means that the initialization process (see `intro`(2)) inherits each of these processes.
- Each attached shared memory segment is detached and the value of `shm_nattch` in the data structure associated with its shared memory identifier is decremented by 1.
- For each semaphore for which the calling process has set a `semadj` value (see `semop`(2)), that `semadj` value is added to the *semval* for the specified semaphore.
- If the process has a process lock, an `unlock` is performed (see `plock`(2)).
- If the process ID, tty group ID, and process group ID of the calling process are equal, the `SIGHUP` signal is sent to each process that has a process group ID equal to that of the calling process.
- If the calling process is the last process in a session to exit, then all nonpersistent IPC facilities created by processes in the session will be removed as if an `IPC_RMID` had been performed on the facility (see `msgget`(2), `semget`(2), and `shmget`(2)).

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

On Cray MPP systems, three additional types of exit are available:

`globalexit` or `_globalexit`	Forces all PEs into global exit processing, which cleans up all resources in the PEs allocated by the application. The call also causes the Cray PVP system's process to exit. `globalexit` also executes library clean up routines on the local PE before calling `_globalexit`.
`localexit` or `_localexit`	Terminates all threads on the local PE. The last PE to call `_localexit` also initiates a `_globalexit` system call. `_localexit` does not destroy the address space of the application on the calling PE. `localexit` also executes library clean up routines on the local PE before calling `_localexit`.

`_threadexit`	Terminates the calling thread, either the main user thread or a user protocol thread. The last thread on the PE to call `_threadexit` also initiates a `_localexit` system call.

The `_exit` system call is equivalent to `_localexit`, and `exit` is equivalent to `localexit`. `exit` does library cleanup on the local PE and then calls `_exit`.

NOTES

See the NOTES section on the `signal`(2) man page.

In UNICOS 9.0, a new process model is introduced for multitasked applications. Instead of a multitasked application being considered as multiple processes, an application will be treated as a single process that includes multiple *light-weight processes* (LWPs). Calling `exit` or `_exit` from a multitasking program terminates the entire multitasking group instead of just the calling LWP.

The old behavior for `_exit` will continue to be available through the `_lwp_exit` system call; this is not recommended for general use. `_lwp_exit` is intended only for use by system software.

The `_newexit` and `newexit` system calls were provided to give early access to the new behavior of `_exit` and `exit` in UNICOS 8.0. These will be removed in a subsequent UNICOS release.

FORTRAN EXTENSIONS

The `exit` system call can be called from Fortran as a subroutine:

```
CALL EXIT ([istat])
```

The `newexit` system call can be called from Fortran as a subroutine (on all systems except Cray MPP systems):

```
CALL NEWEXIT ([istat])
```

The *istat* argument is optional integer exit status. If none is specified, the exit status is 0.

FILES

`/usr/include/sys/stdlib.h`	Contains C prototypes for the `exit`, `newexit`, `globalexit`, and `localexit` system calls
`/usr/include/unistd.h`	Contains C prototypes for the `_exit`, `_newexit`, `_lwp_exit`, `_globalexit`, `_localexit`, and `_threadexit` system calls

SEE ALSO

`intro`(2), `msgget`(2), `plock`(2), `semget`(2), `semop`(2), `shmget`(2), `signal`(2), `wait`(2)

`t_exit`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

`fcntl` – Controls open files

SYNOPSIS

```
#include <fcntl.h>
int fcntl (int fildes, int cmd, int arg);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `fcntl` system call provides control over open files as specified by the following arguments:

fildes Specifies an open file descriptor. It is obtained from a `creat`(2), `dup`(2), `fcntl`, `open`(2), or `pipe`(2) system call.

cmd Specifies an action for `fcntl` to perform. The action can be one of the following:

`F_DUPFD` Returns a new file descriptor, as follows:

Lowest-numbered available file descriptor greater than or equal to *arg*.

Same open file (or pipe) as the original file.

Same file pointer as the original file (that is, both file descriptors share one file pointer).

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

Same file status flags (that is, both file descriptors share the same file status flags).

The Close-on-exec flag associated with the new file descriptor is set to remain open across `exec`(2) system calls.

`F_GETFD` Gets the Close-on-exec flag associated with the *fildes* file descriptor. If the low-order bit is 0, the file remains open across `exec`; otherwise, the file is Closed on execution of `exec`.

`F_SETFD` Sets the Close-on-exec flag associated with *fildes* to the low-order bit of *arg* (0 or 1, as stated previously).

`F_GETFL` Gets file status flags.

`F_SETFL` Sets file status flags to *arg*. Only the `O_APPEND`, `O_NDELAY`, `O_NONBLOCK`, `O_RAW`, and `O_T3D` flags can be set; see `open`(2), `read`(2), and `write`(2).

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

F_SETLK Sets or clears a file segment lock according to the variable of type `struct flock` pointed to by *arg*. The *cmd* F_SETLK establishes read (F_RDLCK) and write (F_WRLCK) locks and removes either type of lock (F_UNLCK). If a read or write lock cannot be set, `fcntl` will return immediately with an error value of –1.

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

F_SETSB Sets the `sitebits` field in the inode for *fildes* to the value specified in *arg*. The file system type of the file referred to by *fildes* must be NC1FS.

F_SETALF Adds the allocation flag bits specified by the value of `arg` to any existing allocation flag bits. Valid allocation flag bits are S_ALF_NOGROW and S_ALF_PARTR (see `stat.h`).

F_CLRALF Removes the allocation flag bits specified by the value of *arg* to any existing allocation flag bits. Valid allocation flag bits are S_ALF_NOGROW and S_ALF_PARTR (see the `stat.h` file).

F_GETXT Gets address extent information from the inode for *fildes*. The calling process must be the owner of the file or have appropriate privilege.

F_RSETLK
F_RSETLKW
F_RGETLK Used by the network lock daemon, `lockd`(8), to communicate with the NFS server kernel to handle locks on NFS files. Specify these options at your own risk. A file lock may be removed if you use them.

arg Specifies a value that varies in meaning according to the action specified by *cmd*.

This number indicates the file descriptor if *cmd* is F_DUPFD, the Close-on-exec flag if *cmd* is F_SETFD, the File Status flag if *cmd* is F_SETFL, the address of a variable of type `struct flock` if *cmd* is F_GETLK, F_SETLK, or F_SETLKW, the one word octal `sitebits` value if *cmd* is F_SETSB, or address of a variable of type `struct xt_report` if *cmd* is F_GETXT.

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

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

The flock structure describes the type (l_type), starting offset (l_whence), relative offset (l_start), size (l_len), process ID (l_pid), and system ID (l_sysid) of the segment of the file to be affected. The process ID and system ID fields are used only with the F_GETLK *cmd* to return the values for a blocking lock. Locks may start and extend beyond the current end of a file but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end of file by setting l_len to 0. If such a lock also has l_whence and l_start set to 0, the whole file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a fork(2) system call.

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

If *cmd* is F_GETLK, F_SETLK, or F_SETLKW and the file is located on a UNICOS shared file system (SFS), *cmd* affects the entire file, and cannot be used to specify areas of the file. If locks are cleared with either F_SETLK or F_SETLKW and F_UNLCK, an O_EXCL open lock is also cleared.

If *cmd* is F_SETSB, *arg* contains the single word sitebits value.

If *cmd* is F_GETXT, *arg* contains the address of the variable of type struct xt_report. The calling process supplies the number of extents to return in the xtr_size field of the *arg* variable. Information returned by fcntl includes the total number of extents (xtr_nextent) for the file, the number of indirect blocks (xtr_nindirs), the number of data blocks (xtr_nblocks), the number of data blocks in primary partitions (xtr_pblocks), the number of data blocks in secondary partitions (xtr_sblocks) of the file system, the minimum (xtr_minblks) and the maximum (xtr_maxblks) data blocks in a single extent, and the data extents (xtr_xtnt) for the file. The data extent fields contain the starting block and number of contiguous blocks, each as a 32-bit field.

NOTES

In the future, the errno variable will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process; therefore, portable application programs should expect and test for either value.

If F_GETLK is specified, the process must be granted read permission to the file via the security label. That is, the active security label of the process must be greater than or equal to the security label of the file.

If F_SETLK, F_SETLKW, F_SETSB, F_SETALF, or F_CLRALF is specified, the process must be granted write permission to the file via the security label. That is, the active security label of the process must equal the security label of the file.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_FOWNER	The process is considered the owner of the file.
PRIV_MAC_READ	The process is granted read permission to the file via the security label.

PRIV_MAC_WRITE The process is granted write permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted read and write permission to the file via the security label and is considered the owner of the file.

RETURN VALUES

If fcntl completes successfully, the value returned depends on the *cmd* argument. The value returned is as follows:

F_DUPFD	A new file descriptor
F_GETFD	Value of flag (only the low-order bit is defined)
F_SETFD	Value other than –1
F_GETFL	Value of file flags
F_SETFL	Value other than –1
F_GETLK	Value other than –1
F_SETLK	Value other than –1
F_SETLKW	Value other than –1
F_SETSB	Value other than –1
F_SETALF	Value of the allocation flags for the file prior to request
F_CLRALF	Value of the allocation flags for the file prior to request
F_GETXT	Value other than –1

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

ERRORS

The fcntl system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	The *cmd* argument is F_SETLK, the type of lock (l_type) is a read (F_RDLCK) lock, and the segment of a file to be locked is already write locked by another process, or the type is a write (F_WRLCK) lock, and the segment of a file to be locked is already read or write locked by another process.
EBADF	The *fildes* argument is not a valid open file descriptor.
EBADF	The requested command is F_GETLK and the calling process does not have MAC read access to the file which the file descriptor refers, or the command was F_SETLK, F_SETLKW, F_SETALF, or F_CLRALF, and the calling process does not have MAC write access to the file to which the file descriptor refers.

EDEADLK	The *cmd* argument is F_SETLKW, the lock is blocked by a lock from another process, and it would cause a deadlock to put the calling process to sleep and wait for that lock to become free.
EFAULT	The *cmd* argument is F_SETLK, or F_GETXT and *arg* points outside the program address space.
EINTR	A signal was caught during the fcntl system call.
EINVAL	The *cmd* argument is F_DUPFD, and *arg* is negative, greater than, or equal to the maximum number of files per process (OPEN_MAX).
EINVAL	The *cmd* argument is F_GETLK, F_SETLK, or SETLKW, and *arg* or the data to which it points is not valid.
EINVAL	The *fildes* argument refers to a file on a foreign file system (for example, NFS).
EINVAL	The *cmd* is F_SETSB, and the *fildes* argument does not refer to a regular file (for example, a directory).
EINVFS	The *fildes* argument refers to a file on a file system that is not a NC1FS file system.
EMFILE	The *cmd* argument is F_DUPFD, and NOFILE file descriptors are currently open.
ENOLCK	The *cmd* argument is F_SETLK or F_SETLKW, the type of lock is a read or write lock, and there is no space for additional record locks to be set (too many file segments locked) because the system maximum has been exceeded.
EPERM	The *cmd* argument is F_GETXT, and the calling process is not the owner of the file and does not have appropriate privilege.

FORTRAN EXTENSIONS

The fcntl system call can be called from Fortran as a function:

```
INTEGER fildes, cmd, arg, FCNTL, I
I = FCNTL (fildes, cmd, arg)
```

Alternatively, fcntl can be called from Fortran as a subroutine. In this case, the return value of the system call is unavailable.

```
INTEGER fildes, cmd, arg
CALL FCNTL (fildes, cmd, arg)
```

The Fortran program must not specify both the subroutine call and the function reference to fcntl from the same procedure.

EXAMPLES

The following examples illustrate different uses of the `fcntl` system call.

Example 1: The file `datafile` is opened for writing only, and the current offset into the file is set to the beginning of the file since the `O_APPEND` flag was not specified on the `open` request.

Later in the program, it is desirable for all writes to append onto the end of the file. After first obtaining the current file status flags using the `F_GETFL` command, the `fcntl` system call sets the `O_APPEND` option with the `F_SETFL` command. All writes to `datafile` extend the file.

```
int fd, flags;

fd = open("datafile", O_WRONLY);

flags = fcntl(fd, F_GETFL, 0);       /* get current file status flags */
fcntl(fd, F_SETFL, flags | O_APPEND);/* add append mode for the file */
```

Example 2: By default, any open file remains open in the new program when an `exec` system call overlays the current program with a new program. (Some system calls in the example are not supported on Cray MPP systems.)

The `myfile` file is open when the `execl` request is made. However, the `fcntl` system call with the `F_SETFD` command sets the `close-on-exec` flag to value `1` before the `execl` request is issued. Therefore, `myfile` is closed when the `execl` request is made, and it does not remain open in the new program named `newprog`.

```
int fd;

fd = open("myfile", O_RDONLY);

fcntl(fd, F_SETFD, 1);     /* file myfile should close on exec(2) */

execl("/tmp/newprog", "newprog", "arg1", "arg2", 0);
```

Example 3: This example shows how `fcntl` safely updates a specific record of a file. The first `fcntl` request sets a write lock on the tenth record of `dbfile`. This record is then read, modified in memory, and rewritten back to the device. The second `fcntl` request unlocks the locked record.

Because `fcntl` uses `F_SETLK`, the request fails immediately if any other process has a lock set that blocks setting this lock.

```
struct flock lock;
int fd, recsize;

fd = open("dbfile", O_RDWR);

lseek(fd, 9 * recsize, 0);/* seek to 10th record in the file */

lock.l_type = F_WRLCK;    /* set write lock */
lock.l_whence = 1;        /* starting offset = current location */
lock.l_start = 0;         /* relative offset = current location */
lock.l_len = recsize;     /* # of bytes to lock - one record */

if (fcntl(fd, F_SETLK, &lock) == -1) {
     perror("record locking operation failed");
     exit(1);
}

/* Code here is to update file record including read and write operations. */

lseek(fd, 9 * recsize, 0);/* seek again to 10th record in the file */

lock.l_type = F_UNLCK;    /* set to unlock */
lock.l_whence = 1;        /* starting offset = current location */
lock.l_start = 0;         /* relative offset = current location */
lock.l_len = recsize;     /* # of bytes to unlock - one record */

if (fcntl(fd, F_SETLK, &lock) == -1) {
     perror("record unlocking operation failed");
     exit(1);
}
```

Example 4: The following `fcntl` system call opens and write locks the entire file (named `datafile`). The `F_SETLKW` command to the `fcntl` request causes the program to sleep if there is any other lock set on the file that causes this lock to be blocked.

When a blocking lock is released, this lock proceeds.

```
struct flock lock;
int fd;

fd = open("datafile", O_RDWR);

lock.l_type = F_WRLCK;      /* set write lock */
lock.l_whence = 1;          /* starting offset = current location */
lock.l_start = 0;           /* relative offset = current location */
lock.l_len = 0;             /* # of bytes to lock - to EOF */

if (fcntl(fd, F_SETLKW, &lock) == -1) {
     perror("record locking operation failed");
     exit(1);
}
```

Example 5: The `fcntl` request with `F_SETLK` command attempts to lock the entire file (named `lockfile`). If this attempt fails, an additional `fcntl` request with command `F_GETLK` determines the process ID of the process that currently holds a lock on the file.

```
int fd;
struct flock lock;

fd = open("lockfile", O_RDONLY);

lock.l_type = F_RDLCK;       /* set read lock */
lock.l_whence = 1;           /* starting offset = current location */
lock.l_start = 0;            /* relative offset = current location */
lock.l_len = 0;              /* # of bytes to lock -> to EOF */
lock.l_pid = 0;              /* initialize pid for later */

if (fcntl(fd, F_SETLK, &lock) == -1) {
     perror("file locking operation failed");
     fcntl(fd, F_GETLK, &lock);/* who's got it locked ? */
     if (lock.l_pid != 0) {
          printf("process having file 'lockfile' locked = %d\n",
               lock.l_pid);
     }
     lock.l_type = F_RDLCK;   /* set read lock again since */
                              /*  F_GETLK resets to F_UNLCK */
}
```

FILES

`/usr/include/fcntl.h`	Contains symbol definitions for the `fcntl` system call
`/usr/include/sys/stat.h`	Contains definitions of `S_ALF_NOGROW` and `S_ALF_PARTR`

SEE ALSO

`chmod`(2), `close`(2), `creat`(2), `dup`(2), `exec`(2), `fork`(2), `open`(2), `pipe`(2), `read`(2), `write`(2)

`fcntl`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`lockd`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS File Formats and Special Files Reference Manual, Cray Research publication SR–2014

NAME

fgetpal – Gets the privilege assignment list (PAL) and privilege sets of a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/priv.h>
int fgetpal (int fdes, pal_t *buf, int bufsize);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The fgetpal system call gets the privilege assignment list (PAL) and privilege sets of the file identified by a file descriptor and returns the information in the buffer.

The fgetpal system call accepts the following arguments:

fdes Specifies file descriptor that identifies the file for which the PAL and privilege sets are retrieved.

buf Contains pointer to the return buffer.

bufsize Indicates the maximum size of the buffer in bytes.

The calling process must have MAC read access to the file or have PRIV_MAC_READ in its effective privilege set.

RETURN VALUES

If fgetpal completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The fgetpal system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	The caller is denied MAC read access to the file.
EBADF	The supplied file descriptor is invalid.
EFAULT	The *buf* argument points outside the address space of the process.
EINVAL	The *bufsize* argument specifies an invalid value.

SEE ALSO

fsetpal(2), getpal(2), setpal(2)

NAME

`fork` – Creates a new process

SYNOPSIS

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

pid_t fork (void);
```

IMPLEMENTATION

Cray PVP systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `fork` system call creates a new process (child process), which is an exact copy of the calling process (parent process). This means that the child process inherits the following attributes from the parent process:

- Environment
- Close-on-exec flag (see `exec`(2))
- Signal handling settings
- Set-user-ID mode bit
- Set-group-ID mode bit
- Profiling on/off status
- *nice* value (see `nice`(2))
- Process group ID
- Job ID
- tty group ID (see `exit`(2) and `signal`(2))
- Trace flag (see `ptrace`(2) *request* `0`)
- Current working directory
- Root directory
- File mode creation mask (see `umask`(2))
- File size limit (see `ulimit`(2))
- All attached shared memory segments (see `shmat`(2))

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

- The child process has a unique process ID.
- The child process has a different parent process ID (that is, the process ID of the parent process).
- The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent process.
- Process locks are not inherited by the child process (see `plock`(2)).
- The `utime`, `stime`, `cutime`, and `cstime` of the child process are set to 0. The time left until an alarm clock signal is reset to 0.
- Record locks set by the parent process are not inherited by the child process (see `fcntl`(2) and `lockf`(3C)).
- In a multitasking group, only the process that executed the `fork` system call is copied.
- Each attached shared memory segment is attached and the value of `shm_nattch` in the data structure associated with the shared memory segment is incremented by 1.
- All `semadj` values are cleared (see `semop`(2))

NOTES

The child process inherits all active and authorized security attributes from the parent process. These attributes include levels, compartments, categories, privileges, and privilege text.

RETURN VALUES

If `fork` completes successfully, it returns a value of 0 to the child process and returns the process ID of the child process to the parent process; otherwise, a value of –1 is returned to the parent process, no child process is created, and `errno` is set to indicate the error.

ERRORS

The `fork` system call fails and no child process is created if one of the following error conditions occurs:

Error Code	Description
`EAGAIN`	The system-imposed limit on the total number of processes under execution in the whole system (`NPROC`) is exceeded.
`EAGAIN`	The system-imposed limit on the total number of processes under execution by one user (`CHILD_MAX`) is exceeded.
`EBUSY`	This error also occurs if you try to enable accounting when it is already enabled or if you issue a `restart`(2) attempt when another job or process in the system is using the *jid* or any *pid* associated with the job (or process) to be restarted.

EINTR	An asynchronous signal (such as `interrupt` or `quit`), which you have elected to catch, occurred during a `fork` system call. When execution resumed after processing the signal, the interrupted system call returned this error condition.
EMEMLIM	More memory space was requested than is allowed for the processes attached to this lnode. The maximum value is set by the `-c` option of the `shradmin`(8) command. This error appears only on systems running the fair-share scheduler.
ENOEXEC	A request was made to execute a file that, although it has the appropriate permissions, does not start with a valid magic number (see `a.out`(5)).
ENOMEM	During an `exec`(2) or `sbreak`(2) system call, a program requested more space than the system could supply. This is not a temporary condition; the maximum space specification is a system parameter.
EPROCLIM	More processes were requested than is allowed for this lnode. The maximum value is set by the `-p` option of the `shradmin`(8) command. This error appears only on systems running the fair-share scheduler.

FORTRAN EXTENSIONS

The `fork` system call can be called from Fortran as a function:

```
INTEGER FORK, I
I = FORK ()
```

EXAMPLES

The following examples illustrate different uses of the `fork` system call.

Example 1: The `fork` request generates a new process that executes the same program as the parent process. The program executing in the parent process that issued the `fork` request is also executing in the child process at the completion of the request. (Usually, in application programs, when a parent generates a child process, the programmer intends for the child process to execute a different program than the one executing in the parent process as illustrated in example 2.)

The return value from `fork` indicates whether execution is in the parent or child process.

```
pid_t res;

if ((res = fork()) == -1) {
     perror("fork failed");
     exit(1);
}

if (res == 0) {
     /* code here is executed in the child process */
}
else {
     /* code here is executed in the parent process */
}
```

Example 2: In many cases, when a parent generates a child process, the programmer wants a different program to execute in the child process rather than the same program as in the parent process. Therefore, when the child process returns from `fork`, it immediately issues an `exec`(2) system call.

In this example, the newly created child process performs an `execl`(2) request to load a different program (`childprog`) for execution. The parent and child processes then execute different programs in parallel.

```
pid_t res;

if ((res = fork()) == -1) {
     perror("fork failed");
     exit(1);
}

if (res == 0) {            /* In child process? */
     execl("childprog", "childprog", "arg1", "arg2", 0);
     perror("exec for childprog failed");
     _exit(1);
}

/* parent process continues execution here */
```

FILES

`/usr/include/sys/types.h`	Contains types required by ANSI X3J11
`/usr/include/unistd.h`	Contains C prototype for the `fork` system call

SEE ALSO

brk(2), exec(2), execl(2), exit(2), fcntl(2), nice(2), plock(2), ptrace(2), restart(2), sbreak(2), semop(2), shmat(2), signal(2), times(2), ulimit(2), umask(2), wait(2)

lockf(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

a.out(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

shradmin(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

fsetpal – Sets the privilege assignment list (PAL) and privilege sets of a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/priv.h>
int fsetpal (int fdes, pal_t *buf, int bufsize);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The fsetpal system call sets the privilege assignment list (PAL) and privilege sets of the file identified by a file descriptor using the information in the buffer.

The fsetpal system call accepts the following arguments:

fdes Specifies file descriptor that identifies the file for which the PAL and privilege sets are set.

buf Contains pointer that contains information.

bufsize Indicates the maximum size of the buffer in bytes.

The calling process must have PRIV_SETFPRIV in its effective privilege set, and must either be the file's owner or have PRIV_FOWNER in its effective privilege set. The calling process must have MAC write access to the file or have PRIV_MAC_WRITE in effective privilege set. The calling process can change the state of privileges in the file's allowed, forced, or set-effective privilege sets only when those privileges are also in the caller's permitted privilege set.

If the PRIV_SU option is enabled, any process with effective user ID 0 meets all the requirements specified in the previous paragraph.

RETURN VALUES

If fsetpal completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The fsetpal system call fails if one of the following error conditions occurs:

Error Code	Description
EBADF	The supplied file descriptor is invalid.
EBADF	The process is not granted MAC write permission to the file and does not have appropriate privilege.

EFAULT	The *buf* argument points outside the address space of the process.
EINVAL	The *bufsize* argument specifies an invalid value.
EINVAL	The contents of the supplied PAL is invalid.
EPERM	The process is not the file owner, and does not have appropriate privilege.
EROFS	The affected file system is a read-only file system.
ESECADM	The process does not have appropriate privilege to use this system call.

SEE ALSO

fgetpal(2), getpal(2), setpal(2)

NAME

fsync – Synchronizes the in-core state of a file with that on disk

SYNOPSIS

```
#include <unistd.h>
int fsync (int fildes);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The fsync system call moves all modified data and attributes of a file descriptor to a permanent storage device. It accepts the following argument:

fildes Specifies the file descriptor.

All in-core modified copies of buffers for the associated file have been written to a disk when the call returns. Programs requiring that a file be in a known state should use this call.

In contrast, the sync(2) system call schedules disk I/O for all files (as if an fsync system call had been done on all files), but returns before the I/O completes.

RETURN VALUES

When fsync completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The fsync system call fails if one of the following error conditions occurs:

Error Code	Description
EBADF	The value of *fildes* is not a valid file descriptor.
EINVAL	File descriptor *fildes* refers to a socket, not a file.
EIO	An I/O error occurred during a read from or a write to the file system.

EXAMPLES

The following example shows how to use the fsync system call to ensure that a device is updated before a file is closed and a process exits. Because the O_RAW flag is not specified in the open(2) request, file datafile is opened by use of the buffered I/O method. When write operations update datafile, these changes are made in system cache buffers and the actual updates to the device are delayed.

Completion of the fsync request assures the user that these changes have reached the device(s) before datafile is closed and the process exits.

```
#include <fcntl.h>

main()
}
     int fd;

     fd = open("datafile", O_RDWR);

     /* updates to file "datafile" performed here */

     if (fsync(fd) == -1) {  /* insure that data arrives on device */
          perror("fsync failed");
          exit(1);
     }

     close(fd);
}
```

FILES

/usr/include/unistd.h	Contains C prototype for the fsync system call

SEE ALSO

open(2), sync(2)

cron(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

getash – Gets an array session handle

SYNOPSIS

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

ash_t getash (void);
```

IMPLEMENTATION

IRIX and UNICOS systems

DESCRIPTION

The getash system call returns the array session handle for the array session that contains the calling process.

The handle for an array session is assigned when the array session is first created. This handle can be overridden using the privileged setash(2) system call.

RETURN VALUES

The getash system call always returns the array session handle. There are no error conditions.

SEE ALSO

newarraysess(2), setash(2)

array_services(7), array_sessions(7)

NAME

getdents – Reads and formats directory entries as file system independent

SYNOPSIS

```
#include <sys/dirent.h>
int getdents (int fildes, char *buf, unsigned nbyte);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getdents system call tries to read a specified number of bytes from a directory and to format them as file system-independent directory entries in a buffer. Because the file system-independent directory entries vary in length, usually the actual number of bytes returned is strictly less than the number of bytes specified.

The getdents system call accepts the following arguments:

fildes Specifies a file descriptor associated with a directory. It is obtained from an open(2) or dup(2) system call.

buf Points to the buffer.

nbyte Specifies the number of bytes.

The file system-independent directory entry is specified by the dirent structure (see dirent(5)).

On devices capable of seeking, getdents starts at a position in the file given by the file pointer associated with *fildes*. On return from getdents, the file pointer is incremented to point to the next directory entry.

This system call was developed to implement the readdir(3C) routine (for a description see directory(3C)), and it should not be used for other purposes.

NOTES

The process must be granted read permission to the file via the security label. That is, the active security label of the process must be greater than or equal to the security label of the file.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_MAC_READ	The process is granted read permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted read permission to the file via the security label.

RETURN VALUES

If `getdents` completes successfully, a nonnegative integer is returned, indicating the number of bytes actually read. A value of 0 indicates that the end of the directory has been reached. If the system call fails, a –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getdents` system call fails if one of the following conditions occurs:

Error Code	Description
`EBADF`	The *fildes* argument is not a valid file descriptor open for reading.
`EBADF`	The active security label of the process is not greater than or equal to the security label of the directory, and the process does not have appropriate privilege.
`EFAULT`	The *buf* argument points outside the allocated address space.
`EINVAL`	The *nbyte* argument is not large enough for one directory entry.
`EIO`	An I/O error occurred while the file system was being accessed.
`ENOENT`	The current file pointer for the directory is not located at a valid entry.
`ENOTDIR`	The *fildes* argument is not a directory.

SEE ALSO

`dup`(2), `open`(2)

`directory`(3C), `readdir`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`dirent`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

getdevn – Gets device number or driver entry

SYNOPSIS

```
int getdevn (unsigned long number, int bflag);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getdevn system call gets a device number or driver entry. It accepts the following arguments:

number Indicates major device number or driver name. If *number* is a major device number, getdevn returns the driver name as a character constant; If *number* is larger than the largest major number, getdevn assumes that it is a character constant and returns the major number for that device.

bflag Indicates device type. If *bflag* is specified (nonzero), the device is assumed to be a block device; if it is 0, the device is a character device.

RETURN VALUES

When getdevn completes successfully, a nonnegative value is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getdevn system call fails if the following error condition occurs:

Error Code	Description
EINVAL	The arguments specify a device number that is undefined or a driver entry that is undefined.

EXAMPLES

The following examples show how to use the getdevn system call.

Example 1: In this example, getdevn returns the major number of the block disk driver:

```
i = getdevn ('dev_dd',1);
```

Example 2: In this example, getdevn returns the major number of the hi.c driver:

```
i = getdevn ('dev_hppi',0);
```

NAME

getfacl – Gets access control list entries for file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/acl.h>
int getfacl (char *fname, struct acl *aclents, int count);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getfacl system call gets the access control list (ACL) entries of a file and stores them in an array.

The getfacl system call accepts the following arguments:

fname Specifies the file that contains ACL entries.

aclents Specifies the array.

count Indicates the maximum number of ACL entries that can be put in the array.

NOTES

Errors are recorded in the security log if discretionary access control logging is enabled.

The maximum number of ACL entries supported is defined by ACLSIZE in acl.h.

The process must be granted read permission to the file via the security label. That is, the active security label of the process must be greater than or equal to the security label of the file.

To be granted search permission to a component of the path prefix, the active security label of the process must be greater than or equal to the security label of the component.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_DAC_OVERRIDE	The process is granted search permission to a component of the path prefix via the permission bits and ACL.
PRIV_MAC_READ	The process is granted search permission to a component of the path prefix via the security label.
PRIV_MAC_READ	The process is granted read permission to the file via the security label.

If the PRIV_SU configuration option is enabled, the super user is granted search permission to every component of the path prefix and is granted read permission to the file via the security label.

RETURN VALUES

If `getfacl` completes successfully, the number of entries in the file's ACL is returned (depending on the value of *count*, this may not necessarily be the number of entries returned in the array *aclents*); otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getfacl` system call fails if one of following error conditions occurs:

Error Code	Description
EACCES	Process does not have mandatory read access to the file.
EFAULT	The *fname* argument points outside the process address space.
EFAULT	The *aclents* argument resides outside the process' address space.
EINVAL	The *count* argument is less than 0.
EMANDV	The process is denied read permission to the file via the security label.
ENAMETOOLONG	The supplied file name is too long.
ENOENT	The specified file does not exist.
ENOTDIR	A component of the path prefix is not a directory.

EXAMPLES

This example shows how to use the `getfacl` system call to retrieve ACL entries for a file. That is, the `getfacl` request retrieves the ACL entries for file `datafile`. Then, these entries are displayed on `stdout`.

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

main(int argc, char *argv[])
{
     struct acl buf[ACLSIZE];
     struct passwd *pwptr;
     int no, i;

     if ((no = getfacl("datafile", buf, ACLSIZE)) == -1) {
          perror("getfacl failed");
          exit(1);
     }

     printf("Access control list for datafile contains ");
     printf("the following users:\n\n");
     printf("ID         Logon ID   Name");
     printf("                        Permissions\n\n");

     for (i = 0; i < no; i++) {
          pwptr = getpwuid(buf[i].ac_usid);
          printf("%-5d      %-10s %-25s   %c%c%c\n",
                  buf[i].ac_usid, pwptr->pw_name, pwptr->pw_gecos,
                  buf[i].ac_mode & 04 ? 'r' : ' ',
                  buf[i].ac_mode & 02 ? 'w' : ' ',
                  buf[i].ac_mode & 01 ? 'x' : ' ');
     }
}
```

FILES

`/usr/include/sys/acl.h`	Defines access control list structure

SEE ALSO

`rmfacl`(2), `secstat`(2), `setfacl`(2)

`spacl`(1), `spclr`(1), `spset`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`slog`(4), `acl`(5), `dirent`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

General UNICOS System Administration, Cray Research publication SG–2301

NAME

getgroups, setgroups – Gets or sets group list

SYNOPSIS

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

int getgroups (int ngroups, gid_t *gidset);

int setgroups (int ngroups, gid_t *gidset);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4 (applies only to getgroups)

DESCRIPTION

The getgroups system call gets the current group list of the user process and stores it in the *gidset* array. It accepts the following arguments:

ngroups Indicates the maximum number of entries that may be placed in *gidset*. No more than NGROUPS_MAX, as defined in the sys/param.h file, are ever returned.

gidset Points to the array.

The setgroups system call sets the group list of the current user process according to the *gidset* array. The *ngroups* argument indicates the number of entries in the array, and it must be no more than NGROUPS_MAX, as defined in the sys/param.h file. Only a process with appropriate privilege can set its group list.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_SETGID	The process is allowed to set its group list.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_ID permbit is allowed to set its group list.

RETURN VALUES

If `getgroups` completes successfully, `getgroups` returns the number of groups to which the user process belongs; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

If `setgroups` completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getgroups` or `setgroups` system call fails if one of the following error conditions occurs:

Error Code	Description
`EFAULT`	The *gidset* argument specifies an address that was not valid.
`EINVAL`	The *ngroups* argument is smaller than the number of groups to which the user process belongs.
`EPERM`	The process does not have appropriate privilege to set its group list.

EXAMPLES

This example shows how to use the `getgroups` system call to retrieve the list of valid groups for the current process. That is, the `getgroups` request retrieves the list for the current process. Then, the list is displayed on `stdout`.

`NGROUPS_MAX`, defined in the `sys/param.h` file, specifies the maximum number of groups in which a user can be a member.

```
gid_t group[NGROUPS_MAX];
int no, i;

if ((no = getgroups(NGROUPS_MAX, group)) == -1) {
     perror("getgroups failed");
     exit(1);
}

printf("The current user belongs to the following groups:\n\n");
for (i = 0; i < no; i++) {
     printf("%d\n", group[i]);
}
```

FILES

`/usr/include/sys/types.h`	Contains types required by ANSI X3J11
`/usr/include/sys/param.h`	Defines configuration files

`/usr/include/unistd.h`	Contains C prototype for the `getgroups` and `setgroups` system calls

SEE ALSO

`initgroups`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

gethostid, sethostid – Gets or sets unique identifier of local host

SYNOPSIS

```
#include <unistd.h>
int gethostid (void);
int sethostid (int hostid);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The gethostid system call returns the 32-bit identifier for the local host.

The sethostid system call establishes a 32-bit identifier, which is intended to be unique among identifiers of all existing systems running UNIX or UNICOS software. This identifier is usually the DARPA Internet address for the local host. Only a process with appropriate privilege can use this call; it is typically performed at boot time.

The sethostid system call accepts the following argument:

hostid Specifies the host identifier.

Before sethostid is called, the default identifier value is 0.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_ADMIN	The process is allowed to set the host ID.

If the PRIV_SU configuration option is enabled, the super user is allowed to set the host ID.

RETURN VALUES

The gethostid system call returns the host ID.

If sethostid completes successfully, 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The `sethostid` system call fails if the following error condition occurs:

Error Code	Description
`EPERM`	The process does not have appropriate privilege to set the host ID.

FILES

`/usr/include/unistd.h`	Contains C prototype for the `gethostid` and `sethostid` system calls

SEE ALSO

`gethostname`(2)

`hostid`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

gethostname, sethostname – Gets or sets name of local host

SYNOPSIS

```
#include <unistd.h>
int gethostname (char *name, int namelen);
int sethostname (char *name, int namelen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The gethostname system call returns the official host name for the local host, as previously set by sethostname. The returned name is null-terminated, unless insufficient space is provided.

The sethostname system call sets the name of the host machine. This call is restricted to a process with appropriate privilege and is normally used only when the network is initialized. Before sethostname is called, the default host name is the null string.

The gethostname and sethostname system calls accept the following arguments:

name Points to the address of an array of bytes where the name is to be stored (gethostname) or is stored (sethostname).

namelen Specifies the size of the *name* array. This length is measured in bytes.

NOTES

If *namelen* is less than the length of the host's official name, the official name is truncated to fit.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_ADMIN	The process is allowed to set the host name.

If the PRIV_SU configuration option is enabled, the super user is allowed to set the host name.

RETURN VALUES

The gethostname system call returns the host name.

If sethostname completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The gethostname or sethostname system call fails if one of the following conditions occurs:

Error Code	Description
EFAULT	For the gethostname or sethostname system call, the *name* or *namelen* argument gave an address that was not valid.
EPERM	The process does not have appropriate privilege to set the host name.

BUGS

For the sethostname system call, host names are limited to MAXHOSTNAMELEN (defined in the /usr/include/sys/param.h file).

EXAMPLES

The following example shows how to use the gethostname system call to retrieve the official host name for the local host. This gethostname request finds the official name of the local host and places it in the array hostname as a null-terminated character string.

```
#define HOSTNM_MAX  256

char hostname[HOSTNM_MAX];

if (gethostname(hostname, HOSTNM_MAX) == -1) {
     perror("gethostname failed");
     exit(1);
}
else {
     /* official host name for local host now
        resides in array hostname */
}
```

FILES

/usr/include/sys/param.h	Defines configuration files
/usr/include/unistd.h	Contains C prototype for the gethostname and sethostname system calls

SEE ALSO

gethostid(2)

hostname(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`getinfo` – Gets specified user, job, or process signal information

SYNOPSIS

```
#include <sys/getinfo.h>
int getinfo (int type, int id, char *arg, int size);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `getinfo` system call obtains detailed information from the kernel about why a `SIGINFO` signal occurred for a process.

type Specifies the type of request. Currently, `GI_SIGINFO` is the only valid *type*.

If *type* = `GI_SIGINFO`, `getinfo` returns a mask. The *id* argument is ignored, the *arg* argument is the address of a `long` integer that will receive the `SIGINFO` mask, and *size* is `sizeof(long)`. The `SIGINFO` mask is defined in `sys/getinfo.h`, and it consists of a bit for every possible reason a `SIGINFO` signal could occur. The bits in the mask are, as follows:

`RESERVED_0`	(1<< 0)	Reserved
`RESERVED_1`	(1<< 1)	Reserved
`SIGINFO_AFQL`	(1<< 2)	Account file quota limit
`SIGINFO_AFQW`	(1<< 3)	Account file quota warning
`SIGINFO_AIQL`	(1<< 4)	Account inode quota limit
`SIGINFO_AIQW`	(1<< 5)	Account inode quota warning
`RESERVED_6`	(1<< 6)	Reserved
`RESERVED_7`	(1<< 7)	Reserved
`SIGINFO_GFQL`	(1<< 8)	Group file quota limit
`SIGINFO_GFQW`	(1<< 9)	Group file quota warning
`SIGINFO_GIQL`	(1<<10)	Group inode quota limit
`SIGINFO_GIQW`	(1<<11)	Group inode quota warning
`RESERVED_12`	(1<< 12)	Reserved
`RESERVED_13`	(1<< 13)	Reserved
`SIGINFO_UFQL`	(1<<14)	User file quota limit
`SIGINFO_UFQW`	(1<<15)	User file quota warning
`SIGINFO_UIQL`	(1<<16)	User inode quota limit
`SIGINFO_UIQW`	(1<<17)	User inode quota warning
`SIGINFO_MIG`	(1<<27)	A file is being migrated online
`SIGINFO_PSLW`	(1<<40)	Process soft limit warning

SIGINFO_SSLW	(1<<41)	Session soft limit warning
SIGINFO_USLW	(1<<42)	User soft limit warning

id Specifies 0. 0 is the only valid value. If specified, the *id* argument is ignored; the request is always performed for the calling process.

arg Points to an argument dependent on *type*.

size Specifies the size of *arg* in characters. This is intended to provide interface compatibility if future enhancements are needed.

NOTES

The active security label of the process must be greater than or equal to the security label of every affected process.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_MAC_READ	The process is allowed to override security label restrictions.
PRIV_POWNER	The process is considered the owner of every affected process.

If the PRIV_SU configuration option is enabled, the super user is considered the owner of every affected process. If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions.

RETURN VALUES

If getinfo completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getinfo system call fails if one of the following error conditions occurs:

Error Code	Description
EFAULT	The *size* argument is smaller than the size needed to return the data.
EFAULT	The *arg* argument points outside the allocated address space.
EINVAL	The *type* argument is an invalid request.

EXAMPLES

UNICOS supports inode and file size quota enforcement. This example shows how to use the `getinfo` system call to determine what type of quota is reached during the execution of a process.

```
#include <signal.h>

main()
{
     void handler(int signo);

     signal(SIGINFO, handler);

     /* If, during execution of this process, an inode or file size
        warning (or limit) level is reached, UNICOS will deliver a
        SIGINFO signal to this process.  By default, the signal is
        ignored by the process.  Since it was chosen to catch a SIGINFO
        signal, if one of these quotas is reached, the process is
        interrupted and the signal handling routine named "handler"
        is executed.  In "handler" the getinfo(2) request is made to
        request more detailed information from the kernel as to why
        the signal was sent (that is, what quota was exceeded).  If
        "handler" does not exit, then processing continues. */

}

void handler(int signo)
{
     long reason;

     printf("A signal of type %d (SIGINFO) was caught.  ", signo);
     printf("See following for reason.\n");
     if (getinfo(GI_SIGINFO, 0, (char *) &reason, sizeof(long)) == -1) {
          perror("getinfo failed");
          exit(1);
     }
     printf("The SIGINFO mask = %o (see getinfo.h for reason)\n", reason);
}
```

SEE ALSO

`quotactl`(2), `signal`(2)

NAME

getjtab – Gets the job table entry associated with a process

SYNOPSIS

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

int getjtab (struct jtab *buf);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getjtab system call gets a copy of the job entry associated with the current process. It accepts the following arguments:

buf Points to the jtab structure. Information concerning the job is placed in this structure.

The jtab structure includes the following members (for a complete list, see /usr/include/sys/jtab.h):

```
int           j_jid;                /* Job id of this entry */
int           j_uid;                /* Job owner user-id */
int           j_ppid;               /* Process-id of job parent */
int           j_signal;             /* Signal notification for job parent */
int           j_nice;               /* Nice value of job */
time_t        j_cpulimit;           /* Max #of cpu clocks allowed */
long          j_cproclimit;         /* Max #of concurrent processes allowed */
long          j_memlimit;           /* Memory size limit (clicks) */
long          j_fsblklimit;         /* Max # of file system blocks */
                                    /* to consume */
long          j_sdslimit;           /* SDS limit in clicks rounded up */
                                    /* to minimum allocation unit; */
                                    /* CRAY Y-MP systems and */
                                    /* Cray MPP systems */
time_t        j_ucputime;           /* Total user cpu time for the job */
time_t        j_scputime;           /* Total system cpu time for the job */
long          j_nprocs;             /* Total number of processes active */
long          j_memuse;             /* Memory size in use by job (clicks) */
long          j_memhiwat;           /* Maximum memory ever used */
                                    /* by job (clicks) */
long          j_fsblkused;          /* # of file system blocks consumed */
long          j_sdsuse;             /* SDS in use; CRAY Y-MP systems */
                                    /* and Cray MPP systems */
unsigned char j_tapelimit [J_NTAPES]; /* Tape group limits; enforced */
                                    /* by tape  daemon */
```

RETURN VALUES

If `getjtab` completes successfully, the job ID is returned. A job ID of 0 indicates that the process is not part of a job. Otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getjtab` system call fails if the *buf* argument points to an address that was not valid, `EFAULT`.

EXAMPLES

This `getjtab` example shows how a process can retrieve all of the information about a job associated with it:

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

main()
{
     struct jtab jdata;
     int jid;

     if ((jid = getjtab(&jdata)) == 0) {
          fprintf(stderr, "This process is not part of a job!\n");
          exit(0);
     }
     else {
          if (jid == -1) {
               perror("getjtab failed");
               exit(1);
          }
     }

     /* information about the current job now available in jdata */
}
```

SEE ALSO

`fork`(2), `intro`(2), `killm`(2), `limit`(2), `nicem`(2), `setjob`(2), `signal`(2), `suspend`(2)

UNICOS File Formats and Special Files Reference Manual, Cray Research publication SR–2014

NAME

`getlim` – Obtains current resource limit information

SYNOPSIS

```
#include <errno.h>
#include <sys/category.h>
#include <sys/resource.h>

int getlim (int id, struct resclim *rptr);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `getlim` system call gets resource limit information from the kernel based on the following arguments:

id Specifies the PID, SID, or UID corresponding to the `resclim` field `resc_category`. A 0 indicates the current PID, SID, or UID.

rptr Points to the `resclim` structure. The `resclim` structure pointed to by *rptr* includes the following members (for a complete list, see `/usr/include/sys/resource.h`):

```
struct resclim {
   int  resc_resource;          /* One of: L_CPU */
   int  resc_category;          /* One of: C_PROC, C_SESS, C_UID, C_SESSPROCS */
   int  resc_type;              /* One of: L_T_ABSOLUTE, L_T_HARD, L_T_SOFT */
   int  resc_action;            /* One of: L_A_TERMINATE, L_A_CHECKPOINT */
   long resc_used;              /* Current amount of resource used */
   long resc_value[R_NLIMTYPES];/* Current resource limit value */
                                /* for each of: */
                                /* L_T_ABSOLUTE, L_T_HARD, L_T_SOFT */
};
```

The `resclim` structure fields `resc_resource` and `resc_category` must be set in order to return limit values. The `resc_resource` field represents the resource to be queried. Currently, only CPU resources are supported; therefore, the value of `resc_resource` must be `L_CPU`. The `resc_category` identifies which category of resource to be queried. Acceptable values are: `C_PROC`, `C_SESS`, `C_UID`, and `C_SESSPROCS`. A short description follows:

Value	Description
`C_PROC`	Returns process limits
`C_SESS`	Returns session limits
`C_UID`	Returns user limits
`C_SESSPROCS`	Returns default process limits for the session

The resclim field resc_category determines whether the *id* argument is a PID, SID, or UID. The resc_category of C_SESSPROCS requires an SID.

If the call succeeds, getlim fills in the missing information in the resclim structure. This includes the following fields:

Field	Description
resc_action	Returns a value of L_A_TERMINATE or L_A_CHECKPOINT. This value determines whether when a hard limit is reached the process is checkpointed before termination.
resc_used	Returns the amount of resource currently accumulated at the time of the call. For L_CPU, this value is the amount of CPU clocks accumulated.
resc_value[R_NLIMTYPES]	Returns three values. The resc_value[L_T_ABSOLUTE] field is the absolute resource limit. The resc_value[L_T_HARD] field is the hard resource limit and resc_value[L_T_SOFT] is the soft resource limit. All CPU resource limits are returned in clocks.

NOTES

The active security label of the process must be greater than or equal to the security label of every affected process.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_MAC_READ	The process is allowed to override security label restrictions.
PRIV_POWNER	The process is considered the owner of every affected process.

If the PRIV_SU configuration option is enabled, the super user is considered the owner of every affected process. The super user is allowed to override security label restrictions.

RETURN VALUES

If getlim completes successfully, a value of 0 is returned, and the resclim structure is filled in with appropriate returned values. Otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getlim system call fails and no information is updated in the resclim structure if one of the following conditions occurs:

Error Code	Description
EFAULT	The address specified for *rptr* was not valid.
EINVAL	One of the arguments contains a value that was not valid.

EPERM	The process does not own every affected process and does not have appropriate privilege.
ESRCH	No processes were found that matched the request.
ESRCH	The active security label of the process is not greater than or equal to the security label of every affected process, and the process does not have appropriate privilege.

SEE ALSO

setlim(2)

nlimit(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

nlimit(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

`getmount` – Returns information about the kernel mount table

SYNOPSIS

```
#include <sys/mount.h>
int getmount (struct mntentinfo *mountcopy, struct kmountinfo *mountinfo);
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The `getmount` system call returns general information about the kernel mount table (number of mounted file systems, last time the kernel mount table has been changed, and so on) and information about each mounted file system (file system name, mount point, type, and options). The `getmount` system call returns all or only part of the information, depending on the value of the following arguments:

mountcopy Points to a set of `mntentinfo` structures that contains information about each file system.

mountinfo Points to the structure that contains the general information about the kernel mount table.

The `mntentinfo` structure is defined in the `sys/mount.h` include file, as follows:

```
struct mntentinfo {
     char *fname;          /* file system name */
     char *dir;            /* mount directory */
     char *opts;           /* mount options */
     char *type;           /* file system type */
};
```

The `kmountinfo` structure is defined in the `sys/mount.h` header file, as follows:

```
struct kmountinfo {
    int                nbent;      /* number of mounted file systems */
    long               lastchge;   /* last change in the mount table */
    int                needupdate; /* need to update memory space */
    struct mount       *rootenv;   /* root in the current environment */
    struct mountlength *len;       /* length of the character arrays */
};
```

If you need only general information about the kernel mount table, you can call `getmount` with the following arguments:

```
struct kmountinfo mountinfo;
getmount(NULL, &mountinfo);
```

In this case, only `nbent` and `lastchge` are valid in the `kmountinfo` structure.

If you need information about each mounted file system, use the `setmntent`(3C) and `getmntinfo`(3C) library routines. These routines use the `getmount` system call to access the kernel mount table. You should avoid using `getmount` directly. However, if it becomes necessary to use it, you must perform the following steps:

1. Allocate memory space for the set of `mountlength` structures `len`, declared as follows:

```
struct mntentinfo {
    int fslen;   /*file system length */
    int dirlen;  /* mount directory length */
    int optlen;  /* options length */
};
```

   Ensure that enough space is available for each entry in the kernel mount table.

2. Call `getmount` to get the general information about the kernel mount table, as follows:

```
getmount(NULL, &mountinfo);
```

   At this point, *mountinfo* should contain the number of mounted file systems and the space needed for the file system name, the mount point, and the options of each mounted file system.

3. Allocate memory space for the set of `mntentinfo` structures `mountcopy`.

4. Call the `getmount` system call by using a pointer to the allocated memory:

```
getmount(mountcopy, &mountinfo);
```

   The system call returns the requested information in the set of `mntentinfo` structures (`mountcopy`).

RETURN VALUES

If `getmount` completes successfully, 0 is returned; otherwise, –1 is returned, and `errno` is set to `EFAULT` to indicate the error.

EXAMPLES

The following example illustrates use of the `getmount` system call. The program gets information about each mounted file system from the kernel mount table and prints it.

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

#include <stdio.h>
#include <unistd.h>

main()
   {
   int i, nb, nmnt;
   struct kmountinfo mountinfo= {0, 0, 0, NULL, NULL};
   struct mntentinfo *mountcopy;

   /* Allocate enough memory for the set of mountlength structures */
   nmnt = sysconf(_SC_CRAY_NMOUNT);
   mountinfo.len = (struct mountlength *)
               malloc(nmnt*sizeof(struct mountlength));

   /* get the general information about the kernel mount table */
   getmount(NULL, &mountinfo);

   /* allocate enough memory to get the information about each FS */
   nb = mountinfo.nbent;

   mountcopy = (struct mntentinfo *)malloc(nb*sizeof(struct mntentinfo));

   for (i = 0; i < nb; i++) {
            mountcopy[i].fname = (char *)malloc(mountinfo.len[i].fslen + 1);
            mountcopy[i].dir = (char *)malloc(mountinfo.len[i].dirlen + 1);
            mountcopy[i].opts = (char *)malloc(mountinfo.len[i].optlen + 1);
            mountcopy[i].type = (char *)malloc(FSTYPSZ + 1);
}

/* get the information about each mounted file system */
getmount(mountcopy, &mountinfo);

/* print it */
for (i = 0; i < nb; i++) {
        fprintf(stdout, "file system name: %s\n", mountcopy[i].fname);
        fprintf(stdout, "mount point: %s\n", mountcopy[i].dir);
        fprintf(stdout, "mount options: %s\n", mountcopy[i].opts);
        fprintf(stdout, "file system type: %s\n", mountcopy[i].type);
    }
}
```

FILES

`/usr/include/sys/mount.h`	Contains C prototype for the `getmount` system call

SEE ALSO

`getmntinfo`(3C), `setmntent`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`mount`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

getpal – Gets the privilege assignment list (PAL) and privilege sets of a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/priv.h>
int getpal (char *path, pal_t *buf, int bufsize);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getpal system call gets the privilege assignment list (PAL) and privilege sets of a file and returns the information in a buffer.

The getpal system call accepts the following arguments:

path Points to the file for which the PAL and privilege sets are retrieved.

buf Specifies the return buffer.

bufsize Indicates the maximum size of the buffer in bytes.

The calling process must have MAC read access to the file or have PRIV_MAC_READ in its effective privilege set.

RETURN VALUES

If getpal completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getpal system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	A component of the *path* prefix denies search permission.
EACCES	The caller is denied MAC read access to the file.
EFAULT	The *buf* or *path* argument points outside the address space of the process.
EINVAL	The *bufsize* argument specifies an invalid value.
ENAMETOOLONG	The supplied file name is too long.
ENOENT	The specified file does not exist.

SEE ALSO

fgetpal(2), fsetpal(2), setpal(2)

NAME

getpeername – Gets name of connected peer

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int getpeername (int s, struct sockaddr *name, int *namelen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getpeername system call returns the name (*name*) of the peer connected to socket (*s*). You must initialize the *namelen* argument to indicate the amount of space to which *name* points. On return, *namelen* contains the actual number of bytes in the name returned. If the buffer that is provided is too small, the name is truncated.

The getpeername system call accepts the following arguments:

s Identifies the socket for which the address is desired.

name Points to the address of a sockaddr structure that receives the address of the remote socket connected to *s*.

namelen Points to an integer that receives the length of the address placed in *name*.

NOTES

If the SOCKET_MAC option is enabled, the active security label of the process must be greater than or equal to the security label of the socket. Note that SOCKET_MAC is part of TCP/IP configurable feature variables list in uts/cf/Nmakefile. For more information, see the connect(2) man page.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_MAC_READ	The process is allowed to override the security label restrictions when the SOCKET_MAC option is enabled.

If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions when the SOCKET_MAC option is enabled.

If *s* is an Internet-domain socket, the `sin_addr` and `sin_port` fields of *name* identify only the socket at the other end of the connection and not the remote process or remote user. Additional knowledge is required to interpret those fields. For example, if the `sin_addr` designates another UNICOS system, a `sin_port` value of less than 1024 indicates a connection with trusted software (for example, `rlogin`(1B)), which may include additional identity information in its protocol data stream. If it is necessary to identify the actual user associated with the socket, the communicating peers must agree in advance on a method, such as the sender placing its `sin_port` value in a protected file accessed through NFS (or other means) by the receiver.

Because no sender name information can be obtained from a UNIX-domain socket, the other end of the connection cannot be identified except to the extent that additional authentication techniques are used. Although there are no identity-based access controls that restrict use of `connect`(2) or `sendto`(2) for a UNIX-domain socket, such a socket can be created in a directory to which execute (search) access is restricted. This limits the ability of other processes to connect to the socket. Alternatively, the listening process could place a random value or secret password in a protected file and require that to be included in all messages it accepts; this ensures that only users with access to that file can send valid messages. For both Internet-domain and UNIX-domain, this authentication requires explicit action on the part of the receiver.

RETURN VALUES

If `getpeername` completes successfully, a value of 0 is returned; otherwise, –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getpeername` system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	If the `SOCKET_MAC` option is enabled, the process does not meet the security label requirements and does not have appropriate privilege.
EBADF	Descriptor *s* is not valid.
EFAULT	Argument *name* points to memory that is not in a valid part of the process address space.
ENOBUFS	Insufficient system resources were available to perform the operation.
ENOTCONN	Socket is not connected.
ENOTSOCK	Descriptor *s* is not a socket.

FILES

`/usr/include/sys/socket.h`	Contains the header file for sockets
`/usr/include/sys/types.h`	Contains the header file for types

SEE ALSO

`bind`(2), `connect`(2), `getsockname`(2), `sendto`(2), `socket`(2)

`rlogin`(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

getpermit, setpermit – Gets or sets user permissions

SYNOPSIS

```
#include <unistd.h>
#include <sys/perm.h>
#include <sys/category.h>

int getpermit (long *mask);

int setpermit (int cat, long *mask);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getpermit system call allows a user program to determine the current permissions associated with the current process. The current permissions are returned in the word indicated by *mask*.

The setpermit system call allows a user program to set the permissions associated with the current process or job. Child processes inherit permissions from their parents. Any process can reduce its active permissions, but only an appropriately privileged process can increase the active permissions.

The getpermit and setpermit system calls accept the following arguments:

mask Specifies the word that contains the current permissions.

cat Determines whether the permissions go into effect for the current process (C_PROC) or the entire job (C_JOB).

The following is a list of possible permissions from the user database (UDB) (in file /usr/include/udb.h), descriptions, and possible permissions from the kernel (in file /usr/include/sys/perm.h). The UDB permission bit must be used when comparing to the udb structure, and kernel permissions must be used when using the system calls. For more complete descriptions of the permbits, see libudb(3C).

UDB Permissions	Description	Kernel Permissions
PERMBITS_ACCT	Accounting permission	PERM_ACCT
PERMBITS_ACCTID	Allows any account ID (newacct(1))	none
PERMBITS_ASKACID	Query for active acid	none
PERMBITS_BYPASSLABEL	Bypasses label tape processing	none
PERMBITS_CHOWN	Allows chown(2), chgrp(1), chmod(2)	PERM_CHOWN
PERMBITS_CHROOT	Allows chroot(2) permission	PERM_CHROOT
PERMBITS_DEDIC	Dedicate CPU permission	PERM_DEDIC

UDB Permissions	Description	Kernel Permissions
PERMBITS_DEVMAINT	Allows device diagnostic	PERM_DIAG
PERMBITS_GROUPADM	Group administrator	none
PERMBITS_GUARD	Driver DONUT guard mode	PERM_GUARD
PERMBITS_GUEST	Allows use of guest	PERM_GUEST
PERMBITS_GUESTADM	Guest administrator	PERM_GUESTADM
PERMBITS_ID	Allows ID changes	PERM_ID
PERMBITS_IPCPERSIST	Allows persistent IPC	PERM_IPCPERSIST
PERMBITS_MKNOD	Allows mknod(2) flag	PERM_MKNOD
PERMBITS_MLSMNT	Allows secure file system mount	Unused. This permbit is available to assign to user accounts, but it no longer grants special abilities.
PERMBITS_MOUNT	Allows mount(2)	PERM_MOUNT
PERMBITS_NICE	Allows nice(2) negative values	PERM_NICE
PERMBITS_NOBATCH	Disables batch logins	none
PERMBITS_NOIACTIVE	Disables interactive logins	none
PERMBITS_PLOCK	Allows plock(2)	PERM_PLOCK
PERMBITS_REALTIME	Allows real-time execution permission	PERM_RTIME
PERMBITS_RESLIM	Resource limits permission	PERM_RESC
PERMBITS_RESTRICTED	System login restricted (udbrstrict(8))	none
PERMBITS_SIGANY	Sends signals to anyone	PERM_SIG
PERMBITS_SUSPRES	Suspends/resumes outside job allowed	PERM_SUSPRES
PERMBITS_SYSPARAM	Sets system parameters permission	PERM_SYSP
PERMBITS_TAPEMANAGER	Allows special tape requests	PERM_TAPE
PERMBITS_WRUNLABEL	R/W unlabeled tapes	none
PERMBITS_YP	Yellow pages flag	none

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_RESOURCE	The process is allowed to increase its active permissions.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_RESLIM permbit is allowed to increase its active permissions.

RETURN VALUES

When `getpermit` or `setpermit` completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getpermit` or `setpermit` system call fails if one of the following error conditions occurs:

Error Code	Description
`EFAULT`	The *mask* argument points to memory that is not in a valid part of the process address space.
`EPERM`	The process does not have appropriate privilege to increase its active permissions.

EXAMPLES

This example shows how to use the `getpermit` system call to retrieve user permissions. The `getpermit` request retrieves the user permissions enabled for the current process and places them in the long integer named *mask*.

A bit conversion table in header file `udb.h` simplifies the interpretation of these permissions, which are represented as single bits in the return value placed in *mask*. In this example, the table is used to convert each permission bit represented in *mask* to a meaningful character string for display.

```
/* These must be defined to use the udb conversion tables and
   their definitions must precede the "#include <udb.h>" */

#define UDB_BIT_CONVERSION  1
#define UDB_BIT_TABLE       1

#include <sys/types.h>
#include <udb.h>
#include <unistd.h>

main()
{
     long mask, permit, i;
     int j;

     if (getpermit(&mask) == -1) {
          perror("getpermit failed");
          exit(1);
     }

     if (mask == 0) {
          printf("\nNo permits currently enabled for this process!\n\n");
     }
     else {
```

```
                printf("\nCurrent permits enabled for this process:\n");
                for (i = 1L; i > 0; i <<= 1) {
                     if (permit = mask & i) {
                          for (j = 0; perm_def[j].pmask != 0; j++) {
                               if (perm_def[j].pmask == permit) {
                                    printf("     %s\n", perm_def[j].pname);
                                    break;
                               }
                          }
                     }
                }
           }
      }
```

FILES

`/usr/include/unistd.h`	Contains C prototype for the `getpermit` and `setpermit` system calls

SEE ALSO

`chmod`(2), `mknod`(2), `mount`(2), `nice`(2), `plock`(2)

`chgrp`(1), `login`(1), `newacct`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`libudb`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`cpu`(4), `udb`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`acct`(8), `csanqs`(8), `udbrstrict`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

getpid, getpgrp, getppid, _getlwpid, _getlwppid, newgetpid, newgetppid – Gets process, process group, or parent process IDs

SYNOPSIS

All Cray Research systems:

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

pid_t getpid (void);

pid_t getpgrp (void);

pid_t getppid (void);
```

Cray PVP systems:

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

pid_t _getlwpid (void);

pid_t _getlwppid (void);

pid_t newgetpid (void);

pid_t newgetppid (void);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4 (applies only to getpid, getpgrp, getppid)

DESCRIPTION

The following system calls obtain the process IDs of the calling process:

- The getpid and newgetpid system calls return the process ID.
- The getpgrp system call returns the process group ID.
- The getppid and newgetppid system calls return the parent process ID.
- The _getlwpid system call returns the light-weight process ID.
- The _getlwppid system call returns the light-weight process ID of the process that created the caller's process.

NOTES

The process ID interfaces have changed in UNICOS 9.0 to support a new multitasking model. Whereas previous UNICOS releases support multiple process IDs in a multitasking group, the new model supports a single process ID. In this new model, what were previously called process IDs are called *light-weight process IDs*.

The `_getlwpid` and `_getlwppid` system calls are not recommended for general use; they are provided for use by system software.

The `newgetpid` and `newgetppid` system calls will be removed in a subsequent UNICOS release.

FORTRAN EXTENSIONS

The `getpid` system call can be called from Fortran as a function:

```
INTEGER GETPID, I
I = GETPID ( )
```

The `getpgrp` system call can be called from Fortran as a function:

```
INTEGER GETPGRP, I
I = GETPGRP ( )
```

The `getppid` system call can be called from Fortran as a function:

```
INTEGER GETPPID, I
I = GETPPID ( )
```

The `newgetpid` system call can be called from Fortran as a function (on all systems except Cray MPP systems):

```
INTEGER NEWGETPID, I
I = NEWGETPID ( )
```

The `newgetppid` system call can be called from Fortran as a function (on all systems except Cray MPP systems):

```
INTEGER NEWGETPPID, I
I = NEWGETPPID ( )
```

FILES

`/usr/include/sys/types.h`	Contains types required by ANSI X3J11
`/usr/include/unistd.h`	Contains C prototype for the `getpid`, `getpgrp`, `getppid`, `_getlwpid`, `_getlwppid`, `newgetpid`, and `newgetppid` system calls

SEE ALSO

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

NAME

getppriv – Gets the privilege state of the calling process

SYNOPSIS

```
#include <sys/types.h>
#include <sys/priv.h>
int getppriv (priv_proc_t *buf, int bufsize);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getppriv system call gets the privilege state of the calling process and places it in a buffer.

The getppriv system call accepts the following arguments:

buf Points to the return buffer.

bufsize Indicates the maximum size of the buffer in bytes.

RETURN VALUES

If getppriv completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getppriv system call fails if the following error condition occurs:

Error Code	Description
EFAULT	The *buf* argument points outside the address space of the process.

SEE ALSO

setppriv(2)

NAME

getsectab – Gets security names and associated values

SYNOPSIS

```
#include <sys/sectab.h>
int getsectab (int type, struct sectab *buf);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getsectab system call returns security names and associated values.

The getsectab system call accepts the following arguments:

type Specifies the type of names and values to be returned. The following are valid values:

Value	Description
0	Returns security compartment names and bit mask values.
1	Returns permission names and bit mask values.
2	Returns category names and bit mask values.
3	Returns security flags and bit mask values.
4	Returns security level names and numbers.
6	Returns privilege names and bit mask values.

buf Points to the sectab structure in which the values are returned. A list of names (maximum 64) and a list of values (–1 terminated) are returned.

RETURN VALUES

If getsectab completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getsectab system call fails if one of the following error conditions occurs:

Error Code	Description
EFAULT	The *buf* argument points outside the process address space.
EINVAL	The *type* argument is less than 0 or greater than 5.

EXAMPLES

The following example shows how to use the `getsectab` system call to retrieve all of the security names and their associated values. For each type of name to be retrieved, `getsectab` is called once. The names and associated values are then displayed on `stdout`.

```
#include <sys/sectab.h>
#include <string.h>

main()
{
     static char *names[] = {"Compartment", "Permission", "Integrity category",
                             "Flag", "Security level", "Integrity class",
                             "Privilege"};
     struct sectab sectab;
     int i, j;

     for (i = MINTAB; i <= MAXTAB; i++) {
          if (getsectab(i, &sectab) == -1) {
              perror("getsectab failed");
              exit(1);
          }

          printf("%s names and values(octal):\n\n", names[i]);
          for (j = 0; j < MAXNAMES; j++) {
              if (strlen(sectab.tb_name[j]) == 0 || sectab.tb_num[j] == -1) {
                   continue;       /* ignore null table entries */
              }
              printf("%-25s      %21o\n", sectab.tb_name[j], sectab.tb_num[j]);
          }
          printf("\n");
     }
}
```

FILES

`/usr/include/sys/sectab.h`	Defines structure in which to return security name and value information

SEE ALSO

`secbits`(3C), `secnames`(3C), `secnums`(3C), `secwords`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`sectab`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

getsockname – Gets socket name

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockname (int s, struct sockaddr *name, int *namelen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getsockname system call returns the current name (*name*) of the specified socket (*s*). You must initialize the *namelen* argument to indicate the amount of space to which *name* points. On return, *namelen* contains the actual number of bytes in the name returned.

The getsockname system call accepts the following arguments:

s Identifies the socket.

name Points to the current name of the socket.

namelen Points to the size of the *name* array.

NOTES

If the SOCKET_MAC option is enabled, the active security label of the process must be greater than or equal to the security label of the socket. Note that SOCKET_MAC is part of TCP/IP configurable feature variables list in uts/cf/Nmakefile. For more information, see the connect(2) man page.

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_MAC_READ	The process is allowed to override the security label restrictions when the SOCKET_MAC option is enabled.

If the PRIV_SU configuration option is enabled, the super user is allowed to override security label restrictions when the SOCKET_MAC option is enabled.

RETURN VALUES

If getsockname completes successfully, a value of 0 is returned; otherwise, a value of −1 is returned, and errno is set to indicate the error.

ERRORS

The `getsockname` system call fails if one of the following conditions occurs:

Error Code	Description
`EACCES`	If the `SOCKET_MAC` option is enabled, the process does not meet the security label requirements and does not have appropriate privilege.
`EBADF`	Descriptor *s* is not valid.
`EFAULT`	Argument *name* or *namelen* points to memory that is not in a valid part of the process address space.
`ENOBUFS`	Insufficient system resources were available to perform the operation.
`ENOTSOCK`	Descriptor *s* is not a socket.

FILES

`/usr/include/sys/socket.h`	Contains the header file for sockets
`/usr/include/sys/types.h`	Contains the header file for types

SEE ALSO

`bind`(2), `socket`(2)

NAME

getsockopt, setsockopt – Gets or sets options on sockets

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockopt (int s, int level, int optname, char *optval, int *optlen);
int setsockopt (int s, int level, int optname, char *optval, int optlen);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getsockopt and setsockopt system calls manipulate options associated with a socket. Options can exist at multiple protocol levels; they are always present at the uppermost (socket) level.

The *getsockopt* system call allows an application to request information about a socket. The setsockopt system call allows an application to manipulate options associated with a socket.

The getsockopt(2) and setsockopt(2) accept the following arguments:

s Specifies the descriptor for the socket.

level Specifies the level at which the option resides. To manipulate options at the socket level, specify *level* as SOL_SOCKET. To manipulate options at any other level, supply the protocol number of the protocol that controls the option (for example, to indicate that an option is interpreted by the Transmission Control Protocol (TCP) protocol, set *level* to the protocol number of TCP). For more information, see the getprot(3C) man page.

optname Specifies the name of the option to examine or retrieve. For the list of available options, see the Options subsection.

optval For getsockopt(2), identifies a buffer in which the current values for the requested options are returned.

For setsockopt(2), identifies a buffer in which the new values for the specified options are retrieved.

If no option value is supplied or returned, *optval* can be supplied as 0.

optlen For getsockopt(2), initially contains the size of the buffer to which *optval* points and is modified on return to indicate the actual size of the value returned. It is a value-result argument.

For setsockopt(2), specifies the length of the new values residing in the buffer to which *optval* points.

The *optname* argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The `sys/socket.h` include file contains definitions for socket-level options (see `socket`(2)). Options at other protocol levels vary in format and name (see the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014).

Most socket-level options take an `int` type value for *optval*. For `setsockopt`, the value must be nonzero to enable a Boolean option, or 0 if the option will be disabled. The `SO_LINGER` option uses a `struct linger` value, defined in `sys/socket.h`, which specifies the desired state of the option and the linger interval.

Options

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

Option	Description
`SO_BROADCAST`	Toggles permission to transmit broadcast messages.
`SO_DEBUG`	Toggles recording of debugging information. This option enables debugging in the underlying protocol modules.
`SO_REUSEADDR`	Toggles local address reuse. This option indicates that the rules used in validating addresses supplied in a `bind`(2) system call should allow reuse of local addresses.
`SO_REUSEPORT`	Allows multiple processes to completely duplicate bindings, if all processes set `SO_REUSEPORT` before binding the port. Every process must specify this option, including the first process to use the port. This option permits multiple instances of a program to receive UDP/IP multicast or to broadcast datagrams for the bound port.
`SO_OWNPORT`	Allows a process to bind to a port and prevent other applications from binding to the same port. This option has been improved to provide better defined addresses.
`SO_KEEPALIVE`	Toggles keep connections alive. This option enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken, and processes using the socket are notified through a `SIGPIPE` signal.
`SO_DONTROUTE`	Toggles routing bypass for outgoing messages. This option 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	Lingers on close if data is present. This option controls the action taken when unsent messages are queued on socket, and a close(2) system call is performed. If the socket promises reliable delivery of data, and SO_LINGER is set, the system blocks the process on the close attempt until it can transmit the data or until it decides it cannot deliver the information (a time-out period, termed the *linger interval*, is specified in the setsockopt call when SO_LINGER is requested). If SO_LINGER is disabled, and a close(2) system call is issued, the system processes the close operation in a manner that allows the process to continue as quickly as possible.
SO_SNDBUF	Sets buffer size for output.
SO_RCVBUF	Sets buffer size for input. SO_SNDBUF and SO_RCVBUF are options that can be used 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 limits on these values. There is a system-wide maximum size for an individual socket buffer. The system silently sets the socket buffer size to the system-wide limit if the request exceeds the limit. This limit is configurable through the netvar(8) command. There is also a per-session cumulative limit for all of the sockets used by a session. The setsockopt call returns ELIMIT and leaves the current value unchanged if the new size would exceed the session limit. This limit is configurable through the user's udb entry.
SO_TYPE	Gets the type of the socket (get only). This option is used only with getsockopt. It returns the type of the socket (for example, SOCK_STREAM); it is useful for servers that inherit sockets on startup.
SO_ERROR	Gets and clears error on the socket (get only). This option is used only with getsockopt. It returns any pending error on the socket and clears the error status. It can be used to check for asynchronous errors on connected datagram sockets (type SOCK_DGRAM) or for other asynchronous errors. Values for SO_ERROR correspond to values for errno.
SO_USELOOPBACK	Bypasses hardware when possible.
SO_SNDLOWAT	Sends low-water mark.
SO_RCVLOWAT	Receives low-water mark.
SO_SNDTIMEO	Sends time-out.
SO_RCVTIMEO	Receives time-out.
SO_DOBINLINE	Leaves received out-of-band data in line.

NOTES

If the SOCKET_MAC option is enabled, to set options on a socket, the active security label of the process must equal the security label of the socket. The SOCKET_MAC option is part of TCP/IP configurable feature variables list in uts/cf/Nmakefile. For more information, see the connect(2) man page.

If the SOCKET_MAC option is enabled, to get options on a socket, the active security label of the process must be greater than or equal to the security label of the socket.

A process with the effective privileges shown is granted the following abilities:

Privilege	Description
PRIV_MAC_READ	When the SOCKET_MAC option is enabled, the process is allowed to override the security label restrictions when getting options of a socket.
PRIV_MAC_WRITE	When the SOCKET_MAC option is enabled, the process is allowed to override the security label restrictions when setting options on a socket.

If the PRIV_SU configuration option is enabled, the super user is allowed to override security label requirements when the SOCKET_MAC option is enabled.

RETURN VALUES

If getsockopt or setsockopt completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The getsockopt or setsockopt system call fails if one of the following error conditions occurs:

Error Code	Description
EACCES	If the SOCKET_MAC option is enabled, the process does not meet the security label requirements and does not have the appropriate privilege.
EBADF	The *s* descriptor is not valid.
EFAULT	The options are not in a valid part of the process address space.
EINVAL	The option or level specified is not valid.
ELIMIT	The request would exceed the session's limit.
ENOBUFS	No buffer space is available.
ENOPROTOOPT	The option is unknown; therefore, it has not been obtained (for getsockopt) or set (for setsockopt).
ENOTSOCK	The *s* descriptor is not a socket.

FILES

`/usr/include/sys/socket.h`	Contains the header file for sockets
`/usr/include/sys/types.h`	Contains the header file for types

SEE ALSO

`bind`(2), `close`(2), `socket`(2)

`getprot`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`ip`(4P), `tcp`(4P) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`netvar`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS File Formats and Special Files Reference Manual, Cray Research publication SR–2014

NAME

getsysv – Gets security attributes

SYNOPSIS

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

int getsysv (struct sysv *buf, int bufsize);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getsysv system call returns security attributes. Any process may issue a getsysv request.

The getsysv system call accepts the following arguments:

buf Points to a sysv structure in which information is returned.

bufsize Specifies the size of the sysv structure in bytes.

The sysv structure includes the following members:

```
short   sy_minlvl;              /* minimum security level */
short   sy_maxlvl;              /* maximum security level */
long    sy_valcmp;              /* authorized compartments */
int     sy_slgbufsize;          /* size (in bytes) of /dev/slog */
char    sy_secure_console[24];  /* secure console: administrator (not used) */
char    sy_admin_console[24];   /* default console: administrator
                                   (must be /dev/console) */
char    sy_oper_console[24];    /* secure console: operator (not used) */
int     sy_spare1;              /* filler */
uint    sy_dev_enforce   :  1,  /* Device labeling enforcement */
uint    sy_spare2        : 22,  /* filler */
        sy_ranpass_flag  :  1,  /* machine passwords enabled if set */
        sy_ranpass_min   :  4,  /* machine passwords minimum size */
        sy_ranpass_max   :  4,  /* machine passwords maximum size */
        sy_netw_options  : 32;  /* network security options */
int     sy_overwrite_count;     /* declassify disk overwrite count */
int     sy_declassify_pattern;  /* declassify disk write pattern */
int     sy_sanitize_pattern;    /* scrub disk write pattern */
int     sy_maxlogs;             /* max login attempts before disable */
int     sy_logdelay;            /* delay (seconds) between failed
                                   login attempts */
int     sy_disable_time;        /* duration (seconds) for which a user
                                   is disabled for exceeding maxlogs */
int     sy_delay_mult    : 1;   /* multiply sy_logdelay by the
                                   number of successive failed
                                   login attempts to calculate
                                   the delay time (in seconds) */
int     sy_slg_state     : 1;   /* security log state on/off */
int     sy_slg_discv     : 1;   /* log discretionary violations */
int     sy_slg_mandv     : 1;   /* log mandatory violations */
int     sy_slg_netwv     : 1;   /* log network violations */
int     sy_slg_mkdirv    : 1;   /* log mkdir violations */
int     sy_slg_rmdirv    : 1;   /* log rmdir violations */
int     sy_slg_linkv     : 1;   /* log link violations */
int     sy_slg_all_rm    : 1;   /* log all rm requests */
int     sy_slg_removev   : 1;   /* log rm violations */
int     sy_slg_all_nami  : 1;   /* log all nami requests */
int     sy_slg_all_valid : 1;   /* log all requests */
int     sy_slg_all_netw  : 1;   /* log all network requests */
int     sy_slg_physio_err: 1;   /* log physical I/O errors */
int     sy_slg_path_track: 1;   /* track pathname of all entries */
int     sy_sec_remote    : 1;   /* tcp/ip mand. access control */
int     sy_sec_scrub     : 1;   /* scrub data blocks on delete */
int     sy_nfs_export    : 1;   /* export secure fs via nfs */
```

```
int     sy_nfs_remote    : 1;   /* rw remote fs via nfs */
int     sy_oldtfm        : 1;   /* old style tfm is possible (not used) */
int     sy_stat_restrict : 1;   /* restrict (sec)stat by level */
int     sy_declsfy_disk  : 1;   /* enable declsfy disk action */
int     sy_slg_sulog     : 1;   /* log all su(1) attempts */
int     sy_console_msg   : 1;   /* send a message to the console when
                                   /* MAXLOGS has been exceeded */
int     sy_disable_acct  : 1;   /* disable account when MAXLOGS */
                                   /* exceeded */
int     sy_slg_filexfr   : 1;   /* log all file transfers */
int     sy_slg_nfs       : 1;   /* log CNFS activity */
int     sy_slg_config    : 1;   /* log UNICOS configuration changes */
int     sy_slg_jstart    : 1;   /* log start of job */
int     sy_slg_jend      : 1;   /* log end of job */
int     sy_slg_netcf     : 1;   /* log network configuration changes */
int     sy_slg_suid      : 1;   /* log suid requests */
int     sy_slg_user      : 1;   /* log user name & password for failed
                                   logins */
int     sy_slg_nqscf     : 1;   /* log NQS database changes */
int     sy_slg_nqs       : 1;   /* log NQS activity */
int     sy_slg_trust     : 1;   /* log trusted process activity */
int     sy_mac_command   : 1;   /* perform MAC checks in commands */
int     sy_forced_socket : 1;   /* ON: syslogd uses sockets only */
int     sy_slg_priv      : 1;   /* log use of privilege */
int     sy_audit_chng    : 1;   /* auditing criteria change flag */
int     sy_slg_audit     : 1;   /* log auditing criteria change */
int     sy_slg_chdir     : 1;   /* log chdir requests */
int     sy_slg_crl       : 1;   /* log Cray REEL librarian activity */
int     sy_slg_ipnet     : 1;   /* log IP layer activity */
int     sy_slg_oper      : 1;   /* log operator actions */
int     sy_slg_secsys    : 1;   /* log non_inode security syscalls */
int     sy_slg_shutdown  : 1;   /* log system shutdown */
int     sy_slg_startup   : 1;   /* log system startup */
int     sy_slg_tape      : 1;   /* log tape activity */
int     sy_slg_tchg      : 1;   /* log system time change */
int     sy_slg_dac       : 1;   /* log DAC changes */
int     sy_slg_privileged: 1;   /* privileged to change auditing */
int     sy_fsetid_restrict:1;   /* restrict setuid/setgid file mgmt */
int     sy_secure_pipe   : 1;   /* MAC restricted pipes? */
int     sy_spare3        : 9;   /* filler - use as needed */
int     sy_slg_maxsize;         /* maximum size of security log */
char    sy_slg_dir[PATH_MAX1+1];/* path for active security log */
char    sy_slg_file[PATH_MAX+1];/* filename for active log */
char    sy_slg_fprefix[PATH_MAX+1];/* prefix for retired log */
```

NOTES

The `getsysv` requests are not recorded in the security log.

If *bufsize* is greater than 0, but less than the defined size of a `sysv` structure, the user's buffer is filled in with the amount of data that fits and a successful status is returned.

RETURN VALUES

If `getsysv` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getsysv` system call fails if one of the following error conditions occurs:

Error Code	Description
`EFAULT`	The *buf* argument points outside the process address space.
`EINVAL`	The *bufsize* argument is less than 0. If *bufsize* is greater than the size of the `sysv` structure, *bufsize* is bounded silently by the actual size.

EXAMPLES

The following example shows how to use the `getsysv` system call to retrieve security attributes.

Special handling of the bit status fields in the `sysv` structure is included.

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

main()
{
     struct sysv buf;
     static char *answ[] = {"No", "Yes"};

     if (getsysv(&buf, sizeof(buf)) == -1)  {
          perror("getsysv failed");
          exit(1);
     }

     printf("System minimum security level = %d\n", buf.sy_minlvl);
     printf("System maximum security level = %d\n", buf.sy_maxlvl);
     printf("System authorized compartments= %o\n", buf.sy_valcmp);
     printf("Security log size (/dev/dslog)= %d\n", buf.sy_slgbufsize);

     /* The fields in the sysv structure starting with sy_delay_mult
        are all bit status fields with a 0 value meaning NO (not
        enabled) and a 1 meaning YES (is enabled).  The following
        statements print the status of some of these fields. */

     printf("Log discretionary violations  = %s\n", answ[buf.sy_slg_discv]);
     printf("Log mandatory violations      = %s\n", answ[buf.sy_slg_mandv]);
     printf("Log network violations        = %s\n", answ[buf.sy_slg_netwv]);
     printf("Log mkdir violations          = %s\n", answ[buf.sy_slg_mkdirv]);
}
```

FILES

`/usr/include/sys/param.h`	Defines configuration files
`/usr/include/sys/sysv.h`	Defines structure for system security values
`/usr/include/sys/types.h`	Contains types required by ANSI X3J11

SEE ALSO

`setsysv`(2)

`spset`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

General UNICOS System Administration, Cray Research publication SG–2301

NAME

gettimeofday, settimeofday – Gets or sets date and time

SYNOPSIS

```
#include <sys/time.h>
int gettimeofday (struct timeval *tp, struct timezone *tzp);
int settimeofday (struct timeval *tp, struct timezone *tzp);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The gettimeofday system call gets the system's notion of the current Greenwich time, to microsecond accuracy. The settimeofday system call sets this time. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The accuracy of the system clock is hardware dependent, using the real-time clock.

The gettimeofday and settimeofday system calls accept the following arguments:

tp Points to the timeval structure.

tzp Points to the timezone structure.

The structures to which *tp* and *tzp* point are defined in the sys/time.h file, as follows:

```
struct timeval {
    long   tv_sec;          /* seconds since Jan. 1, 1970 */
    long   tv_usec;         /* and microseconds */
};

struct timezone {
    int    tz_minuteswest;  /* of Greenwich */
    int    tz_dsttime;      /* type of dst correction to apply */
};
```

The timezone structure indicates the local time zone (measured in minutes of time westward from Greenwich) and a flag. A nonzero flag indicates that daylight saving time applies locally during the appropriate part of the year.

If *tzp* is a zero pointer, the time zone information is not returned or set.

An extended kernel timezone structure, kn_timezone, is defined in the sys/time.h file. The gettimeofday and settimeofday system calls may be used to set and retrieve the contents of this structure when *tp* points to a timeval structure that contains -1 in the tv_sec field and the flag value TZ_MAGIC in the tv_usec field. The extended kernel timezone structure is copied to or from the location pointed to by *tzp*. In this case, the value of the time is returned by gettimeofday, but is not modified by settimeofday.

Only a process with appropriate privilege can set the time of day or time zones.

NOTES

A process with the effective privilege shown is granted the following ability:

Privilege	Description
PRIV_TIME	The process is allowed to set the time.

If the PRIV_SU configuration option is enabled, the super user or a process with the PERMBITS_SYSPARAM permbit is allowed to set the time.

The *tzp* argument is compatible only with 4.3 BSD source code. Except in the special case of the extended kernel timezone request described above, the time zone information is silently ignored on a settimeofday request and always returns zeros on a gettimeofday request. Time zone information must always be obtained with the ctime(3C) library routines that honor the TZ environment variable.

RETURN VALUES

If gettimeofday or settimeofday completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and errno is set to indicate the error.

ERRORS

The gettimeofday or settimeofday system call fails if one of the following error conditions occurs:

Error Code	Description
EFAULT	An argument address references memory that is not valid.
EINVAL	The value specified in tv_usec is out-of-range. Valid values are greater or equal to 0 and less than 1 million.
EPERM	The process does not have appropriate privilege to set the time.

EXAMPLES

The following example shows how to use the gettimeofday system call to obtain the time of day accurate to the microsecond:

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

main()
{
     struct timeval tp;
     struct timezone tzp;

     if (gettimeofday(&tp, &tzp) == -1) {
       perror("gettimeofday failed");
       exit(1);
     }

     /* The system time to microsecond accuracy is now available
        in the timeval structure (tp).  Field tp.tv_sec contains
        the number of seconds since January  1, 1970, while
        field tp.tv_usec contains the number of additional
        microseconds. */
}
```

SEE ALSO

`time`(2)

`date`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`ctime`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

getuid, geteuid, getgid, getegid – Gets real user, effective user, real group, or effective group IDs

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid (void);
uid_t geteuid (void);
gid_t getgid (void);
gid_t getegid (void);
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4 (applies only to getuid, geteuid, getgid)

DESCRIPTION

The following system calls obtain the user and group IDs of the calling process:

- The getuid system call returns the real user ID.
- The geteuid system call returns the effective user ID.
- The getgid system call returns the real group ID.
- The getegid system call returns the effective group ID.

FORTRAN EXTENSIONS

The getuid system call can be called from Fortran as a function:

```
INTEGER GETUID, I
I = GETUID ( )
```

The geteuid system call can be called from Fortran as a function:

```
INTEGER GETEUID, I
I = GETEUID ( )
```

The getgid system call can be called from Fortran as a function:

```
INTEGER GETGID, I
I = GETGID ( )
```

The `getegid` system call can be called from Fortran as a function:

```
INTEGER GETEGID, I
I = GETEGID ( )
```

FILES

`/usr/include/sys/types.h`	Contains types required by ANSI X3J11
`/usr/include/unistd.h`	Contains C prototype for the `getuid`, `geteuid`, `getgid`, and `getegid` system calls

SEE ALSO

`intro`(2), `setuid`(2)

NAME

getusrv – Gets security validation attributes of the process

SYNOPSIS

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

int getusrv (struct usrv *buf);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The getusrv system call obtains security validation information for the calling process. It accepts the following argument:

buf Points to a usrv structure in which the information is returned.

A usrv structure includes the following members:

```
short   sv_minlvl;      /* minimum security level */
short   sv_maxlvl;      /* maximum security level */
long    sv_valcmp;      /* authorized compartments */
long    sv_savcmp;      /* TFM_EXEC command saved compartments (not used) */
long    sv_actcmp;      /* active compartments */
short   sv_permit;      /* permissions */
short   sv_actlvl;      /* active security level */
short   sv_savlvl;      /* TFM_EXEC saved security level (not used) */
short   sv_intcls;      /* active integrity class (not used) */
short   sv_maxcls;      /* maximum integrity class (not used) */
long    sv_intcat;      /* active categories */
long    sv_valcat;      /* authorized categories */
struct  {               /* saved integrity parameters over TFM_EXEC
                           (not used) */
        int     actcls  :32;    /* integrity class before TFM_EXEC
                                   (not used) */
        int     actcat  :32;    /* active category before TFM_EXEC
                                   (not used) */
} sv_savint;
int     sv_audit_off    :1;     /* audit on/off flag */
int     sv_audit_chng   :1;     /* audit change flag */
```

NOTES

The `getusrv` requests are not recorded in the security log.

RETURN VALUES

If `getusrv` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

ERRORS

The `getusrv` system call fails if the following error condition occurs:

Error Code	Description
EFAULT	The *buf* argument points outside the address space of the process.

EXAMPLES

The following example shows how to use the `getusrv` system call to retrieve the security attributes for the calling process. This program only displays the user's special permissions. A security administrator (`secadm`) or a trusted process is allowed to see the `usrtrap` flag; this information is not displayed to nonprivileged users.

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

main()
{
     struct usrv buf;

     if (getusrv(&buf) == -1) {
          perror("getusrv failed");
          exit(1);
     }

     /* Security attributes for the calling process now available in the
        usrv structure named buf. */

     printf("My permissions = %o or interpreted as follows:\n", buf.sv_permit);
     if (PERMIT_SUIDGID & buf.sv_permit) printf ("   set-UID or set-GID\n");
     if (PERMIT_USRTRAP & buf.sv_permit) printf ("   user trap mode set\n");
}
```

FILES

`/usr/include/sys/usrv.h`	Contains C prototype for the `getusrv` system call

SEE ALSO

`setucat`(2), `setucmp`(2), `setulvl`(2), `setusrv`(2)

`setucat`(1), `setucmp`(1), `setulvl`(1), `spset`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

General UNICOS System Administration, Cray Research publication SG–2301

NAME

`guestctl` – Controls and reports the status of major guest system functions

SYNOPSIS

```
#include <sys/guest.h>
#include <sys/guestctl.h>

int guestctl (struct gctl *gct);
```

IMPLEMENTATION

Cray PVP systems except CRAY J90 series and CRAY EL series

DESCRIPTION

The `guestctl` (guest control) system call interacts with a kernel to provide guest system status and to support the following requests:

```
#define GCTL_LOCK       1       /* get lock                     */
#define GCTL_UNLOCK     2       /* release lock                 */
#define GCTL_START      3       /* start guest                  */
#define GCTL_STOP       4       /* stop guest                   */
#define GCTL_STAT       5       /* return status                */
#define GCTL_CHANGE     6       /* change CPU allocation        */
#define GCTL_RESID      7       /* reserve id and memory        */
#define GCTL_LOAD       8       /* load guest kernel in memory  */
#define GCTL_FREEZE     9       /* inhibit guest user CPU usage */
#define GCTL_THAW       10      /* permit guest user CPU usage  */
#define GCTL_RELMEM     11      /* release guest memory         */
#define GCTL_RUNLVL     12      /* set system run level (GUEST) */
#define GCTL_RESUME     13      /* resume execution after breakpoint */
#define GCTL_GLOAD      14      /* load data into guest memory  */
```

A `gctl` structure includes the following members:

```
int     gs_id;                  /* kernel id                    */
char    *gs_start;              /* start of kernel in buffer    */
int     gs_ksize;               /* size of kernel in bytes      */
int     gs_psize;               /* size of param file in bytes  */
int     gs_mem;                 /* memory to assign/release     */
int     gs_ttyreq;              /* number of ttys requested     */
char    gs_owner[GS_OWNRSZ];    /* ASCII name of owner          */
int     gs_req;                 /* guestctl request             */
int     gs_status;              /* guestctl reply status        */
int     gs_maxguests;           /* configured number of guests  */
int     gs_grtcnt;              /* maximum # of grt entries     */
int     gs_trace;               /* extended trace flag          */
int     gs_runlvl;              /* system run level             */
int     gs_lockpid;             /* pid of lock owner            */
struct  gdesc gs_gdesc;         /* gdesc buffer                 */
uint    gs_spres  : 1,          /* != 0 if sched params present */
        unused    : 31,
        gs_cpuhog : 8,          /* CPU hog threshold            */
        gs_schint : 8,          /* CPU scheduling interval      */
        gs_winsiz : 8,          /* CPU scheduling window size   */
        gs_minres : 8;          /* minimum CPU residency time   */
word    *gs_gldaddr;            /* guest memory load addr (words)*/
int     gs_gldsize;             /* load size in bytes            */
long    gs_spares[7];           /* unused                       */
struct  gstt gstat[GS_maxguests+1];  /* guest status structure  */
```

A `gstt` structure (embedded in the `gct`) includes the following members:

```
char    gs_name[GS_NAMESZ];     /* system name                  */
long    gs_ttys;                /* assigned ttys                */
int     gs_csim;                /* nonzero if csim              */
int     gs_cpratio;             /* CPU ratio                    */
word    *gs_kba;                /* kernel fwa                   */
word    *gs_kla;                /* kernel lwa                   */
int     gs_memsize;             /* memory size                  */
char    gs_owner[GS_OWNRSZ];    /* ASCII name of owner          */
word    *gs_uba;                /* user fwa                     */
word    *gs_ula;                /* user lwa                     */
int     gs_stt;                 /* system status                */
int     gs_frozen;              /* system frozen flag           */
int     gs_runlvl;              /* system run level             */
int     gs_halt;                /* halt host if guest panics    */
char    gs_panic[GS_PANICBUF];  /* panic buffer contents        */
int     gs_dedclus;             /* use dedicated system cluster */
long    gs_rval;                /* pkt validation disable flags */
long    gs_rovl;                /* res. overlap disable flags   */
long    gs_pkthdl;              /* packet handler flags         */
int     gs_memhole;             /* associated memory hole       */
int     gs_pktvallev;           /* IOS packet validation level  */
uint    gs_dedic  : 1,          /* dedicate requested CPUs      */
        gs_cpureq : 63;         /* number of CPUs requested     */
long    gs_spares[3];           /* unused                       */
```

The status request (`GCTL_STAT`) will return information about the host and any active guest in the `gstat` array of `gstt` structures. The first entry (0) is that of the host. Subsequent entries (0 through `GS_maxguests`) are valid if the allocated memory (`gs_memsize`) is greater than 0. The current status of each valid system entry is returned in `gs_stt` and can be printed using `gstatus[gct->gstat[id].gs_stt]`. Status values include:

```
#define GSS_NONE        0       /* no status                    */
#define GSS_RSRV        1       /* reserving memory & id        */
#define GSS_LOAD        2       /* loading kernel               */
#define GSS_STRT        3       /* starting guest kernel        */
#define GSS_EXEC        4       /* guest kernel is executing    */
#define GSS_STOP        5       /* stopped normally             */
#define GSS_PANC        6       /* stopped due to panic         */
#define GSS_OBST        7       /* stopped but cpu not returned */
#define GSS_GLOAD       8       /* generic guest memory load    */
```

The lock/unlock (GCTL_LOCK/GCTL_UNLOCK) sequence should bracket all of the major guestctl requests. This prevents other users from making conflicting changes to the guest control structures. If the lock is already held (gs_status == EGS_LOCKED), the process id of the lock owner will be returned in gs_lockpid.

A guest system start requires the following five guestctl requests:

1. Lock the guest control structure (GCTL_LOCK)
2. Reserve a guest id and guest memory (GCTL_RESID)*

   Required Fields:

   gs_mem (size in words of requested guest memory)
   gs_ttyreq (number of desired OWS tty connections)

   Optional Fields:

   gs_owner (string representing the system owner)

   Returned if successful:

   gs_id > 0
   gstat[gs_id].gs_memsize > 0 (may be less than requested)

3. Load the guest kernel and binary into memory (GCTL_LOAD)

   Required Fields:

   gs_id (id of guest returned from GCTL_RESID)
   gs_start (local buffer containing kernel and param file)
   gs_ksize (size of kernel in bytes)
   gs_psize (size of parameter file in bytes)

4. Start the guest system (GCTL_START)

   Required Fields:

   gs_id (id of guest returned from GCTL_RESID)

   Optional Fields:

   gstat[gs_id].gs_halt (non-zero for halt on guest panic)
   gs_trace (non-zero to enable additional kernel tracing)

5. Unlock the guest control structure (GCTL_UNLOCK)

*If a guest system has panicked or is stopped and the memory has not yet been released, the id and associated memory may be reused.

To stop a guest system:

1. Lock the guest control structure (GCTL_LOCK)
2. Stop the guest kernel (GCTL_STOP)

   Required Fields:

   gs_id (id of guest returned from GCTL_RESID)

3. Unlock the guest control structure (GCTL_UNLOCK)

To release guest system memory:

1. Lock the guest control structure (GCTL_LOCK)
2. Release the guest memory (GCTL_RELMEM)

   Required Fields:

   gs_id (id of guest returned from GCTL_RESID)

3. Unlock the guest control structure (GCTL_UNLOCK)

The guest change (GCTL_CHANGE) request can be made at any time to update the following information:

```
gs_trace
gstat[id].gs_halt
gstat[id].gs_owner
gs_cpuhog
gs_schint
gs_winsiz
gs_minres
```

A cooperating guest kernel responds to a guestctl freeze request (GCTL_FREEZE) by not scheduling user processes. When a thaw is issued (GCTL_THAW), normal operation resumes.

1. Lock the guest control structure (GCTL_LOCK)
2. Issue the freeze of thaw request (GCTL_FREEZE or GCTL_THAW)

   Required Fields:

   gs_id (id of guest returned from GCTL_RESID)

3. Unlock the guest control structure (GCTL_UNLOCK)

The system run-level (GCTL_RUNLVL) is the only call that can be made from either a host or a guest. It is called by init(8) to inform the host of the general system status (single- or multi-user mode).

Required Fields:

gs_runlvl (GSS_SINGLE_USER or GSS_MULTI_USER)

NOTES

The following privileges are required:

Request	Privilege Required
GCTL_LOCK	PRIV_ADMIN
GCTL_UNLOCK	PRIV_ADMIN
GCTL_START	PRIV_ADMIN
GCTL_STOP	PRIV_ADMIN
GCTL_STAT	(any user)
GCTL_CHANGE	PRIV_ADMIN
GCTL_RESID	PRIV_ADMIN
GCTL_LOAD	PRIV_ADMIN
GCTL_FREEZE	PRIV_ADMIN
GCTL_THAW	PRIV_ADMIN
GCTL_RELMEM	PRIV_ADMIN
GCTL_RUNLVL	PRIV_RESOURCE
GCTL_RESUME	PRIV_ADMIN
GCTL_GLOAD	PRIV_ADMIN

If the PRIV_SU configuration option is enabled, the super user or a user with the PERMBITS_GUEST permbit is allowed to make any of the guestctl requests.

All users are allowed to make the guest status (GCTL_STAT) request.

CAUTIONS

To avoid the unintentional setting or clearing of fields, it is advisable to obtain a current guest status (GCTL_STATUS) to use as input to the change request.

1. Lock the guest control structure (GCTL_LOCK)
2. Obtain a guest status (GCTL_STAT)
3. Edit fields of interest
4. Issue the change request (GCTL_CHANGE)
5. Unlock the guest control structure (GCTL_UNLOCK)

Although available through the standard UNICOS system call interface, the use of this system call for any request except status (GCTL_STAT) is **not** supported. The system call interface may **change without notice**. See the guest(1) man page for information on managing a guest system.

RETURN VALUES

If `guestctl` completes successfully, a value of 0 is returned; otherwise, a value of –1 is returned, and `errno` is set to indicate the error with a feature specific error. The error, which is returned in the `gctl` status word (`gs_status`), is one of the following:

```
EGS_GSOK         0       /* no error                              */
EGS_SLOTRES      1       /* slot reserved error                   */
EGS_NOROOM1      2       /* insufficient memory                   */
EGS_GBOOT        3       /* gboot struct size difference          */
EGS_GINFOLEN     4       /* ginfo struct size difference          */
EGS_GINFONCPU    5       /* CPU config error                      */
EGS_GINFOCLUS    6       /* cluster config error                  */
EGS_NAMEINUSE    7       /* guest name already in use             */
EGS_NOLOCK       8       /* guestctl lock not held                */
EGS_LOCKED       9       /* guestctl locked already               */
EGS_GINFOGSPEC   10      /* gspec struct size difference          */
EGS_OVERLAP      11      /* memory overlap during load            */
EGS_GADDRESS     12      /* g_address out of range                */
EGS_GCLUSTER     13      /* g_cluster out of range                */
EGS_GENTRY       14      /* g_entry out of range                  */
EGS_GGCOM        15      /* g_gcom out of range                   */
EGS_GERRORP      16      /* g_errorp out of range                 */
EGS_GGPI         17      /* g_gpi out of range                    */
EGS_GGSPEC       18      /* g_gspec out of range                  */
EGS_GNAME        19      /* g_name address range error            */
EGS_GPANICBF     20      /* g_panicbf address range error         */
EGS_GPARK        21      /* g_park address range error            */
EGS_NOID         22      /* no available kernel id                */
EGS_NORELACT     23      /* guest memory release error - active   */
EGS_NORELCHM     24      /* guest memory release error - chm()    */
EGS_MEMREQ       25      /* guest memory request error - chm()    */
EGS_GCPIDLE      26      /* g_cpidle out of range                 */
EGS_GCPUSTATE    27      /* g_cpustate out of range               */
EGS_GCPRMSK      28      /* g_cprmsk out of range                 */
EGS_GTRACEMASK   29      /* g_tracemask out of range              */
EGS_GCLUSREQ     30      /* g_gclusreq out of range               */
EGS_GCSSDREQ     31      /* g_gcssdreq out of range               */
EGS_GCLSYS       32      /* g_clsys out of range                  */
EGS_GPSEMAS      33      /* g_psemas out of range                 */
EGS_GCLMASK      34      /* g_clmask out of range                 */
EGS_GCLSPREQ     35      /* g_gclspreq out of range               */
EGS_BADNAME      36      /* guest name has non-printable chars    */
EGS_HOSTID       37      /* operation not supported on host id    */
EGS_GGBOOT       38      /* g_gboot out of range                  */
```

```
EGS_GINFO       39      /* ginfo out of range                   */
EGS_GGSSDREQ    40      /* gssdreq struct size difference       */
EGS_GCPDOWN     41      /* g_cpdown out of range                */
EGS_GMX         42      /* g_gmx out of range                   */
EGS_GMXL        43      /* gmiop struct size difference         */
EGS_GMXN        44      /* I/O cluster configuration error      */
EGS_GPKTCONF    45      /* g_pktconf out of range               */
EGS_GPKTCONFL   46      /* pktconfig struct size difference     */
EGS_NOROOM2     47      /* insufficient memory                  */
EGS_NOROOM3     48      /* insufficient memory                  */
EGS_NOROOM4     49      /* insufficient memory                  */
EGS_GMEXP       50      /* g_gmexp out of range                 */
EGS_GDESC       51      /* gdesc struct size difference         */
EGS_CONREQ      52      /* invalid number of ttys requested     */
EGS_CONREQ1     53      /* insufficient ttys available          */
EGS_WAITIO      54      /* guest I/O in progress                */
EGS_CPU0DOWN    55      /* CPU 0 is down - must be up for boot  */
EGS_NOPERM      56      /* invalid user permissions             */
EGS_NOSUP       57      /* feature not supported on this H/W    */
EGS_GRPXP       58      /* g_grpxp out of range                 */
EGS_GPDDEREC    59      /* g_pdderec out of range               */
EGS_GPDDERECT   60      /* g_pdderect out of range              */
EGS_GCXTAB      61      /* g_gcxtab out of range                */
EGS_GCXSIZE     62      /* g_gcxsize out of range               */
EGS_ADDMEM      63      /* slot not reserved for add memory req.*/
EGS_NOROOM5     64      /* insufficient memory                  */
EGS_NOCPUS      65      /* no CPUs can be assigned to guest     */
EGS_GBOOTWAIT   66      /* g_bootwait out of range              */
EGS_INVALREQ    67      /* invalid or unsupported guestctl() req*/
EGS_ACTLOAD     68      /* cannot load to active system         */
EGS_NOROOM6     69      /* insufficient memory                  */
```

ERRORS

The following UNICOS system errors may map to one or more specific `guestctl` system call errors (`EGS_xxxxx`).

Error Code	Description
`EACCES`	The request (`gs_req`) is not valid on a guest system.
`EAGAIN`	A required resource is temporarily unavailable. See the `guestctl` error for more information.

EBUSY	A release of guest memory cannot yet be performed due to the possibility of outstanding I/O from the previously active guest. See the guestctl error for more information.
EDEADLK	The guestctl is currently held by another process.
EFAULT	A copy of data to or from the user area failed. See the guestctl error for more information.
EINVAL	The request number (gs_req) may not be valid or the specified guest id (gs_id) is greater than MAXGUESTS. See the guestctl error for more information.
ENOMEM	Mainframe memory is not currently available to satisfy the GCTL_RESID request. See the guestctl error for more information.
EPERM	The caller does not have the PRIV_ADMIN privilege. The caller does not have the PERMBITS_GUEST permbit.

FILES

/etc/udb	Contains a list of valid users and lists their privileges and permissions

SEE ALSO

guest(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

init(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022