NAME

intro - introduction to system calls and error numbers

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <errno.h>

DESCRIPTION

This section provides an overview of the system calls, their error returns, and other common definitions and concepts.

RETURN VALUES

Nearly all of the system calls provide an error number referenced via the external identifier errno. This identifier is defined in <*sys/errno.h*> as

```
extern int * __error();
#define errno (* __error())
```

The *__error()* function returns a pointer to a field in the thread specific structure for threads other than the initial thread. For the initial thread and non-threaded processes, *__error()* returns a pointer to a global *errno* variable that is compatible with the previous definition.

When a system call detects an error, it returns an integer value indicating failure (usually -1) and sets the variable *errno* accordingly. (This allows interpretation of the failure on receiving a -1 and to take action accordingly.) Successful calls never set *errno*; once set, it remains until another error occurs. It should only be examined after an error. Note that a number of system calls overload the meanings of these error numbers, and that the meanings must be interpreted according to the type and circumstances of the call.

The following is a complete list of the errors and their names as given in <*sys/errno.h*>.

0 *Undefined error: 0*.Not used.

1 EPERM *Operation not permitted*.An attempt was made to perform an operation limited to processes with appropriate privileges or to the owner of a file or other resources.

2 ENOENT *No such file or directory*.A component of a specified pathname did not exist, or the pathname was an empty string.

3 ESRCH *No such process*.No process could be found corresponding to that specified by the given process ID.

4 EINTR *Interrupted system call*.An asynchronous signal (such as SIGINT or SIGQUIT) was caught by the process during the execution of an interruptible function. If the signal handler performs a normal return, the interrupted system call will seem to have returned the error condition.

5 EIO *Input/output error*.Some physical input or output error occurred. This error will not be reported until a subsequent operation on the same file descriptor and may be lost (over written) by any subsequent errors.

6 ENXIO *Device not configured*.Input or output on a special file referred to a device that did not exist, or made a request beyond the limits of the device. This error may also occur when, for example, a tape drive is not online or no disk pack is loaded on a drive.

7 E2BIG *Argument list too long*.The number of bytes used for the argument and environment list of the new process exceeded the current limit (NCARGS in <*sys/param.h*>).

8 ENOEXEC *Exec format error*.A request was made to execute a file that, although it has the appropriate permissions, was not in the format required for an executable file.

9 EBADF *Bad file descriptor*.A file descriptor argument was out of range, referred to no open file, or a read (write) request was made to a file that was only open for writing (reading).

10 ECHILD *No child processes*.A wait(2) or waitpid(2) function was executed by a process that had no existing or unwaited-for child processes.

11 EDEADLK *Resource deadlock avoided*.An attempt was made to lock a system resource that would have resulted in a deadlock situation.

12 ENOMEM *Cannot allocate memory*.The new process image required more memory than was allowed by the hardware or by system-imposed memory management constraints. A lack of swap space is normally temporary; however, a lack of core is not. Soft limits may be increased to their corresponding hard limits.

13 EACCES *Permission denied*.An attempt was made to access a file in a way forbidden by its file access permissions.

14 EFAULT *Bad address*.The system detected an invalid address in attempting to use an argument of a call.

15 ENOTBLK *Block device required*.A block device operation was attempted on a non-block device or file.

16 EBUSY *Device busy*.An attempt to use a system resource which was in use at the time in a manner which would have conflicted with the request.

17 EEXIST *File exists*.An existing file was mentioned in an inappropriate context, for instance, as the new link name in a link(2) system call.

18 EXDEV *Cross-device link*.A hard link to a file on another file system was attempted.

19 ENODEV *Operation not supported by device*.An attempt was made to apply an inappropriate function to a device, for example, trying to read a write-only device such as a printer.

20 ENOTDIR *Not a directory*.A component of the specified pathname existed, but it was not a directory, when a directory was expected.

21 EISDIR *Is a directory*.An attempt was made to open a directory with write mode specified.

22 EINVAL *Invalid argument*.Some invalid argument was supplied. (For example, specifying an undefined signal to a signal(3) function or a kill(2) system call).

23 ENFILE *Too many open files in system*.Maximum number of open files allowable on the system has been reached and requests for an open cannot be satisfied until at least one has been closed.

24 EMFILE *Too many open files*.Maximum number of file descriptors allowable in the process has been reached and requests for an open cannot be satisfied until at least one has been closed. The getdtablesize(2) system call will obtain the current limit.

25 ENOTTY *Inappropriate ioctl for device*.A control function (see ioctl(2)) was attempted for a file or special device for which the operation was inappropriate.

26 ETXTBSY *Text file busy*.The new process was a pure procedure (shared text) file which was open for writing by another process, or while the pure procedure file was being executed an open(2) call requested write access.

27 EFBIG *File too large*.The size of a file exceeded the maximum.

28 ENOSPC *No space left on device*.A write(2) to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because no more disk blocks were

available on the file system, or the allocation of an inode for a newly created file failed because no more inodes were available on the file system.

29 ESPIPE *Illegal seek*.An lseek(2) system call was issued on a socket, pipe or FIFO.

30 EROFS *Read-only file system*.An attempt was made to modify a file or directory on a file system that was read-only at the time.

31 EMLINK *Too many links*.Maximum allowable hard links to a single file has been exceeded (limit of 32767 hard links per file).

32 EPIPE *Broken pipe*.A write on a pipe, socket or FIFO for which there is no process to read the data.

33 EDOM *Numerical argument out of domain*.A numerical input argument was outside the defined domain of the mathematical function.

34 ERANGE *Result too large*.A numerical result of the function was too large to fit in the available space (perhaps exceeded precision).

35 EAGAIN *Resource temporarily unavailable*.This is a temporary condition and later calls to the same routine may complete normally.

36 EINPROGRESS *Operation now in progress*.An operation that takes a long time to complete (such as a connect(2)) was attempted on a non-blocking object (see fcntl(2)).

37 EALREADY *Operation already in progress*.An operation was attempted on a non-blocking object that already had an operation in progress.

38 ENOTSOCK *Socket operation on non-socket*.Self-explanatory.

39 EDESTADDRREQ *Destination address required*.A required address was omitted from an operation on a socket.

40 EMSGSIZE *Message too long*.A message sent on a socket was larger than the internal message buffer or some other network limit.

41 EPROTOTYPE *Protocol wrong type for socket*.A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type SOCK_STREAM.

42 ENOPROTOOPT *Protocol not available*.A bad option or level was specified in a getsockopt(2) or setsockopt(2) call.

43 EPROTONOSUPPORT *Protocol not supported*.The protocol has not been configured into the system or no implementation for it exists.

44 ESOCKTNOSUPPORT *Socket type not supported*.The support for the socket type has not been configured into the system or no implementation for it exists.

45 EOPNOTSUPP *Operation not supported*.The attempted operation is not supported for the type of object referenced. Usually this occurs when a file descriptor refers to a file or socket that cannot support this operation, for example, trying to *accept* a connection on a datagram socket.

46 EPFNOSUPPORT *Protocol family not supported*.The protocol family has not been configured into the system or no implementation for it exists.

47 EAFNOSUPPORT *Address family not supported by protocol family*.An address incompatible with the requested protocol was used. For example, you should not necessarily expect to be able to use NS addresses with ARPA Internet protocols.

48 EADDRINUSE *Address already in use*.Only one usage of each address is normally permitted.

49 EADDRNOTAVAIL *Can't assign requested address*.Normally results from an attempt to create a socket with an address not on this machine.

50 ENETDOWN *Network is down*.A socket operation encountered a dead network.

51 ENETUNREACH *Network is unreachable*.A socket operation was attempted to an unreachable network.

52 ENETRESET *Network dropped connection on reset*.The host you were connected to crashed and rebooted.

53 ECONNABORTED *Software caused connection abort*.A connection abort was caused internal to your host machine.

54 ECONNRESET *Connection reset by peer*.A connection was forcibly closed by a peer. This normally results from a loss of the connection on the remote socket due to a timeout or a reboot.

55 ENOBUFS *No buffer space available*.An operation on a socket or pipe was not performed because

the system lacked sufficient buffer space or because a queue was full.

56 EISCONN *Socket is already connected*.A connect(2) request was made on an already connected socket; or, a sendto(2) or sendmsg(2) request on a connected socket specified a destination when already connected.

57 ENOTCONN *Socket is not connected*.An request to send or receive data was disallowed because the socket was not connected and (when sending on a datagram socket) no address was supplied.

58 ESHUTDOWN *Can't send after socket shutdown*.A request to send data was disallowed because the socket had already been shut down with a previous shutdown(2) call.

60 ETIMEDOUT *Operation timed out*.A connect(2) or send(2) request failed because the connected party did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.)

61 ECONNREFUSED *Connection refused*.No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.

62 ELOOP *Too many levels of symbolic links*.A path name lookup involved more than 32 (MAXSYMLINKS) symbolic links.

63 ENAMETOOLONG *File name too long*.A component of a path name exceeded {NAME_MAX} characters, or an entire path name exceeded {PATH_MAX} characters. (See also the description of _PC_NO_TRUNC in pathconf(2).)

64 EHOSTDOWN *Host is down*.A socket operation failed because the destination host was down.

65 EHOSTUNREACH *No route to host*.A socket operation was attempted to an unreachable host.

66 ENOTEMPTY *Directory not empty*.A directory with entries other than '.' and '..' was supplied to a remove directory or rename call.

67 EPROCLIM *Too many processes*.

68 EUSERS *Too many users*.The quota system ran out of table entries.

69 EDQUOT *Disc quota exceeded*.A write(2) to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because the user's quota of disk blocks was

exhausted, or the allocation of an inode for a newly created file failed because the user's quota of inodes was exhausted.

70 ESTALE *Stale NFS file handle*.An attempt was made to access an open file (on an NFS file system) which is now unavailable as referenced by the file descriptor. This may indicate the file was deleted on the NFS server or some other catastrophic event occurred.

72 EBADRPC *RPC struct is bad*.Exchange of RPC information was unsuccessful.

73 ERPCMISMATCH *RPC version wrong*.The version of RPC on the remote peer is not compatible with the local version.

74 EPROGUNAVAIL *RPC prog. not avail*.The requested program is not registered on the remote host.

75 EPROGMISMATCH *Program version wrong*.The requested version of the program is not available on the remote host (RPC).

76 EPROCUNAVAIL *Bad procedure for program*.An RPC call was attempted for a procedure which does not exist in the remote program.

77 ENOLCK *No locks available*.A system-imposed limit on the number of simultaneous file locks was reached.

78 ENOSYS *Function not implemented*.Attempted a system call that is not available on this system.

79 EFTYPE *Inappropriate file type or format*.The file was the wrong type for the operation, or a data file had the wrong format.

80 EAUTH *Authentication error*.Attempted to use an invalid authentication ticket to mount a NFS file system.

81 ENEEDAUTH *Need authenticator*.An authentication ticket must be obtained before the given NFS file system may be mounted.

82 EIDRM *Identifier removed*.An IPC identifier was removed while the current process was waiting on it.

83 ENOMSG *No message of desired type*.An IPC message queue does not contain a message of the desired type, or a message catalog does not contain the requested message.

84 EOVERFLOW *Value too large to be stored in data type*.A numerical result of the function was too large to be stored in the caller provided space.

85 ECANCELED *Operation canceled*.The scheduled operation was canceled.

86 EILSEQ *Illegal byte sequence*.While decoding a multibyte character the function came along an invalid or an incomplete sequence of bytes or the given wide character is invalid.

87 ENOATTR *Attribute not found*.The specified extended attribute does not exist.

88 EDOOFUS *Programming error*.A function or API is being abused in a way which could only be detected at run-time.

89 EBADMSG *Bad message*.A corrupted message was detected.

90 EMULTIHOP *Multihop attempted*.This error code is unused, but present for compatibility with other systems.

91 ENOLINK *Link has been severed*.This error code is unused, but present for compatibility with other systems.

92 EPROTO *Protocol error*.A device or socket encountered an unrecoverable protocol error.

93 ENOTCAPABLE *Capabilities insufficient*.An operation on a capability file descriptor requires greater privilege than the capability allows.

94 ECAPMODE *Not permitted in capability mode*.The system call or operation is not permitted for capability mode processes.

95 ENOTRECOVERABLE *State not recoverable*.The state protected by a robust mutex is not recoverable.

96 EOWNERDEAD *Previous owner died*.The owner of a robust mutex terminated while holding the mutex lock.

97 EINTEGRITY *Integrity check failed*.An integrity check such as a check-hash or a cross-correlation failed. The integrity error falls in the kernel I/O stack between EINVAL that identifies errors in parameters to a system call and EIO that identifies errors with the underlying storage media. It is typically raised by intermediate kernel layers such as a filesystem or an in-kernel GEOM subsystem when they detect inconsistencies. Uses include allowing the mount(8) command to

return a different exit value to automate the running of fsck(8) during a system boot.

DEFINITIONS

Process ID.

Each active process in the system is uniquely identified by a non-negative integer called a process ID. The range of this ID is from 0 to 99999.

Parent process ID

A new process is created by a currently active process (see fork(2)). The parent process ID of a process is initially the process ID of its creator. If the creating process exits, the parent process ID of each child is set to the ID of the calling process's reaper (see procctl(2)), normally init(8).

Process Group

Each active process is a member of a process group that is identified by a non-negative integer called the process group ID. This is the process ID of the group leader. This grouping permits the signaling of related processes (see termios(4)) and the job control mechanisms of csh(1).

Session

A session is a set of one or more process groups. A session is created by a successful call to setsid(2), which causes the caller to become the only member of the only process group in the new session.

Session leader

A process that has created a new session by a successful call to setsid(2), is known as a session leader. Only a session leader may acquire a terminal as its controlling terminal (see termios(4)).

Controlling process

A session leader with a controlling terminal is a controlling process.

Controlling terminal

A terminal that is associated with a session is known as the controlling terminal for that session and its members.

Terminal Process Group ID

A terminal may be acquired by a session leader as its controlling terminal. Once a terminal is associated with a session, any of the process groups within the session may be placed into the foreground by setting the terminal process group ID to the ID of the process group. This facility is used to arbitrate between multiple jobs contending for the same terminal; (see csh(1) and tty(4)).

Orphaned Process Group

A process group is considered to be *orphaned* if it is not under the control of a job control shell. More precisely, a process group is orphaned when none of its members has a parent process that is in the same session as the group, but is in a different process group. Note that when a process exits, the parent process for its children is normally changed to be init(8), which is in a separate session. Not all members of an orphaned process group are necessarily orphaned processes (those whose creating process has exited). The process group of a session leader is orphaned by definition.

Real User ID and Real Group ID

Each user on the system is identified by a positive integer termed the real user ID.

Each user is also a member of one or more groups. One of these groups is distinguished from others and used in implementing accounting facilities. The positive integer corresponding to this distinguished group is termed the real group ID.

All processes have a real user ID and real group ID. These are initialized from the equivalent attributes of the process that created it.

Effective User Id, Effective Group Id, and Group Access List

Access to system resources is governed by two values: the effective user ID, and the group access list. The first member of the group access list is also known as the effective group ID. (In POSIX.1, the group access list is known as the set of supplementary group IDs, and it is unspecified whether the effective group ID is a member of the list.)

The effective user ID and effective group ID are initially the process's real user ID and real group ID respectively. Either may be modified through execution of a set-user-ID or set-group-ID file (possibly by one its ancestors) (see execve(2)). By convention, the effective group ID (the first member of the group access list) is duplicated, so that the execution of a set-group-ID program does not result in the loss of the original (real) group ID.

The group access list is a set of group IDs used only in determining resource accessibility. Access checks are performed as described below in ''File Access Permissions''.

Saved Set User ID and Saved Set Group ID

When a process executes a new file, the effective user ID is set to the owner of the file if the file is set-user-ID, and the effective group ID (first element of the group access list) is set to the group of the file if the file is set-group-ID. The effective user ID of the process is then recorded as the saved set-user-ID, and the effective group ID of the process is recorded as the saved set-group-ID. These values may be used to regain those values as the effective user or group ID

after reverting to the real ID (see setuid(2)). (In POSIX.1, the saved set-user-ID and saved set-group-ID are optional, and are used in setuid and setgid, but this does not work as desired for the super-user.)

Super-user

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

Descriptor

An integer assigned by the system when a file is referenced by open(2) or dup(2), or when a socket is created by pipe(2), socket(2) or socketpair(2), which uniquely identifies an access path to that file or socket from a given process or any of its children.

File Name

Names consisting of up to {NAME_MAX} characters may be used to name an ordinary file, special file, or directory.

These characters may be arbitrary eight-bit values, excluding NUL (ASCII 0) and the '/' character (slash, ASCII 47).

Note that it is generally unwise to use '*', '?', '[' or ']' as part of file names because of the special meaning attached to these characters by the shell.

Path Name

A path name is a NUL-terminated character string starting with an optional slash '/', followed by zero or more directory names separated by slashes, optionally followed by a file name. The total length of a path name must be less than {PATH_MAX} characters. (On some systems, this limit may be infinite.)

If a path name begins with a slash, the path search begins at the *root* directory. Otherwise, the search begins from the current working directory. A slash by itself names the root directory. An empty pathname refers to the current directory.

Directory

A directory is a special type of file that contains entries that are references to other files. Directory entries are called links. By convention, a directory contains at least two links, '.' and '..', referred to as *dot* and *dot-dot* respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

Root Directory and Current Working Directory

Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process's root directory need not be the root directory of the root file system.

File Access Permissions

Every file in the file system has a set of access permissions. These permissions are used in determining whether a process may perform a requested operation on the file (such as opening a file for writing). Access permissions are established at the time a file is created. They may be changed at some later time through the chmod(2) call.

File access is broken down according to whether a file may be: read, written, or executed. Directory files use the execute permission to control if the directory may be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the owner of the file, those users in the file's group, anyone else. Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if:

The process's effective user ID is that of the super-user. (Note: even the super-user cannot execute a non-executable file.)

The process's effective user ID matches the user ID of the owner of the file and the owner permissions allow the access.

The process's effective user ID does not match the user ID of the owner of the file, and either the process's effective group ID matches the group ID of the file, or the group ID of the file is in the process's group access list, and the group permissions allow the access.

Neither the effective user ID nor effective group ID and group access list of the process match the corresponding user ID and group ID of the file, but the permissions for ''other users'' allow access.

Otherwise, permission is denied.

Sockets and Address Families

A socket is an endpoint for communication between processes. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties. These properties include whether messages sent and received at a socket require the name of the partner, whether communication is reliable, the format used in naming message recipients, etc.

Each instance of the system supports some collection of socket types; consult socket(2) for more information about the types available and their properties.

Each instance of the system supports some number of sets of communications protocols. Each protocol set supports addresses of a certain format. An Address Family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

SEE ALSO

intro(3), perror(3)