NAME

intro − introduction to system calls

DESCRIPTION

Section 2 of this manual lists all the entries into the system. In most cases two calling sequences are specified, one of which is usable from assembly language, and the other from C. Most of these calls have an error return. From assembly language an erroneous call is always indicated by turning on the c-bit of the condition codes. The presence of an error is most easily tested by the instructions *bes* and *bec* ("branch on error set (or clear)"). These are synonyms for the *bcs* and *bcc* instructions. From C, an error condition is indicated by an otherwise impossible returned value. Almost always this **−1**; the individual sections specify the details.

In both cases an error number is also available. In assembly language, this number is returned in r0 on erroneous calls. From C, the external variable *errno* is set to the error number. *Errno* is not cleared on successful calls, so it should be tested only after an error has occurred. There is a table of messages associated with each error, and a routine for printing the message, see *perror*(3).

The possible error numbers are not recited with each writeup in section 2, since many errors are possible for most of the calls. Here is a list of the error numbers, their names inside the system (for the benefit of system-readers), and the messages available using *perror.* A short explanation is also provided.

Users needing to examine these error codes directly should include the file **/usr/include/errno.h** rather than wiring these numbers into their program.

0 - (unused)

1 **EPERM** Not owner and not super-user
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner. It is also returned for attempts by ordinary users to do things allowed only th the super-user.

2 **ENOENT** No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 **ESRCH** No such process
The process whose number was given to *signal* does not exist or is already dead. Also returned for an attempt to send a message to a process that has not enabled message reception.

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

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

6 **ENXIO** No such device or address
I/O on a special file refers to a subdevice which does not exist, or is beyond the limits of the allowed number of subdevices. It may also occur when, for example, a tape drive is not on-line, or no disk pack is loaded on a drive.

7 **E2BIG** Arg list too long
An argument list longer than 5,120 bytes (counting the null at the end of each argument) is presented to a member of the *exec* family.

8 **ENOEXEC** Exec format error
: a request is made to ececute a file which, 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 (resp. write) request is made to a file which is open only for writing (resp. reading).

10 **ECHILD** No children
: *Wait* was requested but the process has no living or unwaited-for children.

11 **EAGAIN** No more processes
: In a *fork*, the system's process table is full and no more processes can, for the moment, be created.

12 **ENOMEM** Not enough core
: During an *exec* or *break*, a program asks for more memory than the system is able to supply. This is not a temporary condition; the maximum memory size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments is such as to require mor than the existing 8 segmentation registers, or if there is not enough swap space during a *fork*.

13 **EACCES** Permission denied
: An attempt was made to access a file or some other resource in a way forbidden by the protection system.

14 **EFAULT** Memory fault
: User has supplied a non-existent address.

15 **ENOTBLK** Block device required
: A plain file was mentioned where a block device was required, e.g., in *mount*.

16 **EBUSY** Mount device busy
: An attempt to mount a device that was already mounted, or an attempt was made to dismount a device on which there is an open file or which is some process's current directory, or the system profile clock was busy.

17 **EEXIST** File exists
: An existing file was memtioned in an inappropriate context, e.g., *link*.

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

19 **ENODEV** No such device
: An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.

20 **ENOTDIR** Not a directory
: A non-directory was specified where a directory is required, for example in a path name or as an argument to *cd*.

21 **EISDIR** Is a directory
: An attempt to write on a directory.

22 **EINVAL** Invalid argument
: some invalid argument: currently, dismounting a non-mounted device, memtioning an unknown signal in *signal*, giving an unknown request to *ioctl*, passing an invalid argument list to *exec*, providing an unknown *function* argument to *sema* or *msg*, reading or writing a file for which *lseek* has returned a negative pointer, multiple system profiling was requested, or invalid math function arguments (see 3M).

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

24 **EMFILE** Too many open files
Only 20 files can be open per process.

25 **ENOTTY** Not a typewriter
The file mentioned in *ioctl* is not a typewriter or one of the other devices to which these calls apply.

26 **ETXTBSY** Text file busy
An attempt to execute a pure-procedure program which is currently open for writing (or reading). Also an attempt to open for writing a pure- procedure program that is being executed.

27 **EFBIG** File too large
An attempt to make a file larger than the maximum of **ULIMIT**, or 2048 blocks.

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

29 **ESPIPE** Seek on pipe
An *lseek* was issued to a pipe. This error should also be issued for other non-seekable devices.

30 **EROFS** Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 **EMLINK** Too many links
An attempt to make more than 127 links to a file.

32 **EPIPE** Write on broken pipe
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 **ETABLE** No entries left
One of the system tables necessary to complete the request is temporarily full, or the argement of a function in the math package (3M) is out of the domain of the function.

33 **EDOM** Math argument
The argument of a function in the math package (3M) is out of the domain of the function. This error number is used by certain programs that were transported from UNIX/TS to CB-UNIX after that error number was already in use for another purpose in CB-UNIX (see 33 **ETABLE** above).

34 **EFUNC** Invalid function
An attempt to perform an invalid operation, or the value of a function in the math package (3M) is not representable within machine precision.

34 **ERANGE** Result too large
The value of a function in the math package (3M) is not representable within machine precision. This error number is also used for 34 **EFUNC** above. See the note under 33 **EDOM** above.

35 **ENOMSG** No message
A message of the requested type is not on the message queue.

36 **ENOALOC** Resource not allocated
An attempt was made either to use a facility that must first be allocated (e.g., *recvw* without first enabling messages) or th allocate a facility in a way other than for its intended use.

37 **ELOCK** Locking error
An attempt was made to unlock a process or text that was not locked, or an attempt was made to lock the process/text when the text/process was already locked (the locking specifications are mutually exclusive). Other possibilities are that a locked process tried to *exec*, or that *sprofil* could not lock the user buffer in memory.

SEE ALSO

intro (3)

ASSEMBLER

as /usr/include/sys.s file

The PDP-11 assembly language interface is given for each system call. The assembler symbols are defined in **/usr/include/sys.s.**

Return values appear in registers r0 and r1; it is unwise to expect these registers to be preserved. An erroneous call is always indicated by turning on the c-bit of the condition codes. The error number is then returned in r0. The presence of an error is most easily tested by the instructions *bes* and *bec* ("branch on error set (or clear)"). These are synonyms for the *bcs* and *bcc* instructions.

For the *syscb* and *utssys* groups of system calls, the call's number is passed in r1 and the first argument to the call, if there is one, is passed in r0.

OLD C COMPILER

The manual pages in sections 2 and 3 with names ending in ":o" are system calls and functions that are for use with the old C compiler (*occ*) and/or release 1 of CB-UNIX. When using the old C compiler, old system calls are used automatically via special "stamping" of the version number in the load files in the compiled programs (see *stamp*(1)). For system calls that do not exist in the new compiler, the old system routines are called. For system calls that have a *sysent* index identical to a release 2 system call, the stamping determines which will be executed. The ":o" routines are kept around primarily for the commands in section 1 of this manual, which are still compiled with the old C compiler, and for user routines from version 1 systems that were compiled with the old C compiler (which was the new C compiler under the old system). The load files for release 1 user programs have the inherent stamping that will cause the proper system calls and library routines to be executed.

Users should eventually change and recompile programs that require any ":o" routines, so that all programs use the new C compiler and current system calls - the current *occ* will not be available in release 3 of CB-UNIX. Of course, any new programs should be written to use the new C compiler, if at all possible. It is anticipated that the CB-UNIX Systems Group will follow its own advice and change those commands in section 1 that still require the old compiler.