# INTRODUCTION TO SYSTEM CALLS

**GENERAL**

This section lists all the entries into the UNIX Supervisor running under MERT. Not implemented (when compared to UNIX Generic 3) are *acct, chroot, errlog, ioctl,* and *ptrace.* The following calls have changed: *call* (returns after process has been created, not after it has terminated); *fstat, mknod,* and *stat* (different inode format). New are *logname, logdir, logtty,* and *udata.*

**PROGRAMMING FOR PORTABILITY**

To ease conversion to future systems such as DMERT, UNIX/RT, and UNIX/TS, the following rules should be obeyed whenever possible:

1. Use the NEWIO(III) package for input and output.
2. Use *lseek*(III) instead of *seek*(II).
3. When using *fstat*(II), *stat*(II), or *mknod*(II), use new *inode* structure from *include*(V) and compile with *-l7* option.
4. Always obtain system structures from */usr/include* using the *#include* <...> statement, rather than copying them into the program.

MERT Release 0 uses a file system structure similar to UNIX Generic 3 (based on UNIX Version 6 ). File sizes are given as 24-bit quantities (3 bytes). Future versions of UNIX and MERT will offer a new file system structure which greatly increases the file system size and requires a 32-bit (4-byte) size field. The system calls that will be affected are: *stat, fstat, mknod,* and *seek.*

The *stat* and *fstat* calls differ in both format and content of the inode structure returned. The 2-bit file type (060000/bits 12-13) of the flag word has been expanded to 3-bits (070000/bits 12-14). All programs using these calls will have to be changed. You can prepare for this change when writing new programs or adapting existing programs by using the following procedure. Programs using *fstat* or *stat* should contain a statement *#include <stat7.h>* which will obtain the new inode structure for the large file system. Compilations of such programs must currently use the *-l7* option, which will invoke a special library with routines that map the new format into the old inode structure. Once on the new system, no source changes are required, since *stat7.h* will be an alias for the then standard *stat.h* file. (A recompilation - or reloading if object files have been kept - but no source change is all that is required then).

The same holds for *mknod* which must use a new interpretation of the *mode* bits, as given under *file system* (g). *chmod* need not be changed, since it uses only the lower bits which have unchanged meaning. *chowner* can be coded in the new way with 3 arguments, the last of which will be ignored in the current system.

With regard to the *seek* call, two approaches are possible. The simplest and, if you are not using *-l7* recommended, way is to avoid using *seek* alltogether and use *lseek* (III) instead. In the new systems, lseek will be an alias for the new *seek.* Alternatively, use the new *seek* call format (identical to *lseek* with the long offset) and compile or load with the *-l7* option.

ERROR RETURNS

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 is −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 (III).*

The possible error numbers are not recited with each writeup in section II, 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.

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 to 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, attempting 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 dialed in or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 512 bytes (counting the null at the end of each argument) is

presented to *exec*.

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with one of the magic numbers 407 or 410.

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 core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments is such as to require more than the existing 8 segmentation registers.

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 some process's current directory.

17 EEXIST File exists
An existing file was mentioned 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 *chdir*.

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, mentioning an unknown signal in *signal*, giving an unknown request in *stty* to the TIU special file, passing an invalid argument list to *exec*, and providing an unknown *function* argument to *lock* or *msg*.

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

24 EMFILE Too many open files
Only 15 files can be open per process.

25 ENOTTY Not a typewriter
The file mentioned in *stty* or *gtty* 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 32768 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
A *seek* 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 neccessary to complete the request is temporarily full.

34 EFUNC Invalid function
An attempt to perform an invalid operation (e.g. trying to *unlock* a semaphore not locked by the invoking process).

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 to allocate a facility in a way other than for its intended use (e.g. trying to allocate a P-V semaphore for lock-unlock use).

39 EFIRST First access of logical block
(not yet implemented). A block is accessed for reading, that has never been writen nor initialized.

40 ENOMOVE fmove fails
A contiguous file cannot be moved.

41 ENOEXT no extents
A contiguous file has exhausted the maximum number of extents and cannot grow anymore. Copy into larger initial allocation.

42 EPATH pathname too long
There are too many characters in an absolute pathname.