Introduction to UniFLEX™ System Calls

Technical Systems Consultants, Inc.

Introduction to UniFLEX™ System Calls

COPYRIGHT INFORMATION

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

Introduction to UniFLEX System Calls

System Calls

All programs interface to the operating system through system calls. There are a variety of calls which allow file manipulation, task control, and various system functions. The calls are implemented with a SWI3 opcode followed by a one byte function code which defines the call to be performed. Zero to four words (16 bit values) may follow the function code depending on the particular call. The UniFLEX assembler supports the 'sys' pseudo-op which sets up the appropriate machine code for a system call. Its syntax is:

```
        sys    function[,arg0,arg1,arg2,arg3]
```

where function is the system call number or name if using the system call equate file (see below). This pseudo-op produces the SWI3 code for the call, a single byte for the function, and 16 bit values for each argument.

The arguments to system calls fall into three categories. The first is a number. The numbers may be bit patterns such as in the 'chprm' call or mode codes such as in 'open'. In all cases, a 16 bit value is used, even if the number required will fit in 8 bits or less. If the call requires a number larger than 16 bits, such as 'seek' which requires a 32 bit number, the number is passed as two arguments, the first one is the most significant part of the number and the second argument is the least significant. The second type of argument is the pointer. All calls which require a name or ASCII string (such as file names for 'open' and 'create') require a pointer to the name. The pointer is simply the address of the location of the string in memory. The string should always be null terminated (a 00 byte). The final argument type is the buffer address. Several calls, such as 'status', require a place in the caller's address space to place data generated by the call. The argument presented in this case should simply be the 16 bit address pointing to the start of the data buffer to be used. Some calls also extract data from a caller supplied buffer.

Some of the system calls require information to be passed in registers as well as in the form of arguments. Most calls use the D register but a few use X as well. All registers are preserved through a system call unless a value is returned in the register. An error generated in a call always returns the error number in the D register. All condition codes are also preserved through a system call with the exception of the error bit. The error bit is the same as the carry and the assembler supports the mnemonics 'bes' and 'bec' for 'branch if error set' and 'branch if error clear' which are synonymous with 'bcs' and 'bcc' respectively. The error bit will always return cleared if no error resulted from the call, otherwise it will be set and the error response code will be in D. The usage of each system call will be described in a similar manner. An example and explanation follows:

```
<file descriptor in D>
sys read,buffer,count
<bytes read in D>
```

This is the usage for the 'read' system call. The information contained in the angle brackets preceeding the call shows the data expected in the registers by the system. In this example, the D register should contain the file descriptor number of the file to be read. Next is the actual sys call as it would appear in the assembler source listing. The system funcion is 'read' and it has two arguments, 'buffer' and 'count'. Following the call is information regarding the data to be found in the changed registers. In this example, the D register will contain a count which represents the actual number of bytes read from the specified file. All other registers will survive the system call unmodified.

System Errors

As mentioned previously, the system may return from a system call with the error bit set (carry). If this is the case, the D register will contain the number of the resulting error. Following is a list of all system error numbers and their respective meanings.

1 EIO I/O error. This can result from a CRC error, hardware malfunction, or defective media problem while reading or writing a device.

2 EFAULT System fault. System faults are detected by the hardware and vary from system to system.

3 EDTOF Data section overflow. This error can result from a 'break' system call if trying to grow the data section of a program and it overflows into the stack section.

4 ENDR Not a directory. The file name specified is not a directory type file but the system call requires it to be one.

5 EDFUL Device full. The device currently being written has no more available space.

6 ETMFL Too many files. Each task is permitted a maximum of 16 open files at any one time. Attempting to open more than this will produce this error.

7 EBADF Bad file. The file descriptor given does not refer to an open file, or the file mode is not correct for the operation (e.g. file is open for read and a write is attempted).

8 ENOFL No file. The file name specified could not be found and the system call requires the file to exist.

9 EMSDR Missing directory. One of the directory elements specified in a pathname did not exist.

10 EPRM No permission. An attempt was made to perform an action (such as file access) which permission was denied.

11 EFLX File exists. A file name was specified which already existed and the system call requires the file to be previously non-existent.

12 EBARG Bad argument. A bad argument was presented to a system call. This usually implies a number which is out of range or a non-existent mode code.

13 ESEEK Seek error. An attempt was made to seek beyond the beginning of a file or beyond the physically possible maximum size of a file.

14 EXDEV Crossed devices. An attempt was made to link to a file on a different device than the existing file.

15 ENBLK Not a block special file. The file name specified was not a block special file and the system call referenced requires it to be a block device (e.g. mount).

16 EBSY Device busy. The device specified in an 'unmount' is currently being used.

17 ENMNT File not mounted. The file specified to an unmount call was not previously mounted.

18 EBDEV Bad device specified. The system call requires a device type file as an argument.

19 EARGC Too many arguments. Too many arguments were presented to an 'exec' system call and the argument space overflowed. There is an upper limit of approximately 3000 bytes for arguments.

20 EISDR File is a directory. The file specified is a directory and the system call requires it to be a regular type file.

21 ENOTB File is not binary. An attempt was made to execute a file which was not an executable binary file.

22 EBBIG Binary file too big. The binary file specified to 'exec' exceeds the physical address space limits.

23 ESTOF Stack overflow. An attempt was made to grow the stack space which caused it to overflow into the tasks data or text space.

24 ENCHD No children living. A 'wait' system call was executed with no living children tasks to wait for.

25 ETMTS Too many tasks active. An attempt was made to fork a new task which exceeded the systems maximum allowable limit. This error will also result if the system task table becomes full.

26 EBDCL Bad system call. A system call function code was encountered which does not represent an existing system call.

27 EINTR Interrupted system call. One of the program interrupts wich the current task was catching occured during the system call.

28 ENTSK No task found. The task id referenced in the system call did not represent an active task in the system.

29 ENTTY Not a tty. The system call ('ttyget' or 'ttyset') requires the specified file to represent a tty type device.

30 EPIPE Write to broken pipe. An attempt to write data to a pipe which does not have an active read channel open.

31 ELOCK Record lock error. The specified record can not be locked by this task. This usually implies that another task has the requested record locked (or part of the record).

System Definitions

There are several files containing system definitions which reside in the system directory '/lib'. These files should be used as 'library' files in the assembler whenever the appropriate definitions are required. A description of each file follows.

sysdef System call definitions. All of the system call names are defined in this file. All programs written should include this file.

syserrors System errors. All standard system error names and their equated error numbers appear in this file.

sysstat File status block. This file contains the block definition for the information returned by the 'status' and 'ofstat' system calls.

systim Time buffer definitions. The 'time' and 'ttime' system calls return their information in a caller provided buffer. These buffers are defined in this file.

systty TTY buffer. The 'ttyget' and 'ttyset' require a buffer for their data transferal. The contents of this buffer is defined here.

sysints System program interrupts. All program interrupt names are equated to their respective numbers in this file.

alarm

USAGE

```
<seconds in D>
sys alarm
<previous seconds in D>
```

DESCRIPTION

Alarm will cause an alarm interrupt to be issued after the number of seconds specified. At alarm time, the program interrupt ALRMI will be sent to the task. Unless this interrupt is caught or ignored, it will terminate the task. This system call returns immediately to the caller after execution.

DIAGNOSTICS

No errors are possible from this call.

break

USAGE

```
sys break,address
```

DESCRIPTION

Break changes the amount of memory associated with the task. The 'address' specifies the highest address to be used by the task for data. If the address specified is already in the assigned data space, any memory beyond it will be released back to the system.

DIAGNOSTICS

If more memory is requested than is physically possible on the system, an error will be issued.

cdata

USAGE

```
sys cdata,address
```

DESCRIPTION

Cdata is similar to 'break' in that it assigns memory to the task's data space. New memory obtained through cdata will always be physically contiguous which is not the case with break. The 'address' in the call specifies the highest address to be used by the task for data. If the address is already in the data space, the call has no effect. This call is available to allow character type devices which need to access large buffers in memory to do so, without the necessity of memory mapping on the device's controller.

DIAGNOSTICS

If the amount of contiguous memory requested can't be granted, an error will result.

USAGE

```
  sys chacc,fname,perm
```

DESCRIPTION

Chacc is used to check the accessibility of file 'fname'. The 'perm' argument should be '1' for read check, '2' for write check, or '4' for execute check. Any combination of these may be used (e.g. 3 checks read/write). If 'perm' is 0, checks if the directories leading to the file may be searched and if the file actually exists.

DIAGNOSTICS

An error is returned if the file does not exist, if the directory path cannot be searched, or if the permission is not granted.

chdir

USAGE

```
sys chdir,dirname
```

DESCRIPTION

This call is used to change the current user directory to that specified by 'dirname', which points to the actual name. The caller must have execute permission in the specified directory.

DIAGNOSTICS

An error will be issued if the name specified is not a directory, or cannot be searched.

chown

USAGE

```
  sys chown,fname,ownerid
```

DESCRIPTION

Chown will change the owner of the file name pointed at by 'fname'. Ownerid is a 16 bit user-id value. Only the system manager may execute this call.

DIAGNOSTICS

An error is returned if the caller is not the system manager.

chprm

USAGE

```
  sys chprm,fname,perm
```

DESCRIPTION

Chprm will change the access permission bits associated with the file name pointed at by 'fname'. The new permission bits 'perm' will replace the old. The allowable permissions are as follows:

```
* permissions

 FACUR => %00000001 ($01)  owner read permission
 FACUW => %00000010 ($02)  owner write permission
 FACUE => %00000100 ($04)  owner execute permission
 FACOR => %00001000 ($08)  others read permission
 FACOW => %00010000 ($10)  others write permission
 FACOE => %00100000 ($20)  others execute permission
 FXSET => %01000000 ($40)  set id bit for execute
```

DIAGNOSTICS

It is an error if the file does not exist, or the caller is not the file owner or system manager.

close

USAGE

```
<file descriptor in D>
sys close
```

DESCRIPTION

Close the file represented by the file descriptor specified. Files are automatically closed on task termination but it is wise to close them manually whenever possible.

DIAGNOSTICS

An error is returned if the file descriptor is not valid, or if the file has already been closed

cpint

USAGE

```
sys cpint,interrupt,address
<old address in D>
```

DESCRIPTION

Inform the system as to what action it should take on receipt of the 'interrupt' specified. If address is 0, the default action will occur (usually task termination). If the address is 1, the interrupt will be ignored. Any other address will be taken to be a valid user program address where control should be passed upon interrupt interception. After interception, the interrupt number will be in the D register. The user's code should exit the interrupt code via an RTI instruction. Following the return, the task will continue at the point it was interrupted. After processing an intercepted interrupt, the system resets it back to the default condition, therefore, to continue catching the interrupt, it is necessary to re-issue a new 'cpint' call each time the interrupt occurs. It should be noted that the KILLI interrupt cannot be ignored or caught. All interrupts retain their status after a 'fork' but 'exec' resets all caught interrupts back to their default state. The system calls for 'read' and 'write' when referencing a slow device (like a terminal), and the calls 'stop' and 'wait' may return prematurely if a caught interrupt occurs during the system's handling of them. If this happens, it will look as if the system call returned an error (EINTR), and the call can be re-issued if desired. Following is a list of system interrupts. Those marked with '*' cause a core dump if not caught or ignored.

* system interrupts

HANGI	=>	1	hangup interrupt
INTI	=>	2	keyboard interrupt
QUITI	=>	3*	quit interrupt
EMTI	=>	4*	emt interrupt (swi)
KILLI	=>	5	task kill interrupt
WPIPI	=>	6	write broken pipe interrupt
BARGI	=>	7*	bad argument interrupt
TRACI	=>	8*	trace interrupt
TIMEI	=>	9*	time limit interrupt
ALRMI	=>	10	alarm interrupt
TERMI	=>	11	task termination interrupt

DIAGNOSTICS

An error is issued if the interrupt specified is out of range.

create

USAGE

```
  sys create,fname,perm
  <file descriptor in D>
```

DESCRIPTION

Create a new file with access permissions as specified in 'perm'. The permissions are the same as in the 'chprm' call and are as follows:

* permissions

```
 FACUR => %00000001 ($01)  owner read permission
 FACUW => %00000010 ($02)  owner write permission
 FACUE => %00000100 ($04)  owner execute permission
 FACOR => %00001000 ($08)  others read permission
 FACOW => %00010000 ($10)  others write permission
 FACOE => %00100000 ($20)  others execute permission
```

If the file already exists, its length will be truncated to zero (all data deleted) but the original permissions and owner will be retained. In either case, the file is ultimately opened for writing. It is not necessary to specify write permission even though the file will ultimately be opened for write. This allows a task to create a file and disallow others from writing the file until it has been completed.

DESCRIPTION

An error will be issued if there are too many files open, if the files path can not be searched, or if the directory it resides in can not be written.

crpipe

USAGE

```
sys crpipe
<read file descriptor in D>
<write file descriptor in X>
```

DESCRIPTION

This call is used to create pipe for inter-task communication. This call should be used before a 'fork' operation to allow the output of the original task to be used as input by the forked task. Up to 4096 bytes of output may be written into the pipe before the task will be suspended. Once the task doing the reading has read all of the data written, the writing task will again be run. If the writing task closes the file (file descriptor from X) and the reading task consumes all of the data, an end of file condition will result.

DIAGNOSTICS

An error if too many files are opened.

crtsd

USAGE

```
  sys crtsd,fname,desc,address
```

DESCRIPTION

This call is used to create a special file (device) or a new directory. The name of the new file will be 'fname' and it will have the type and permissions as stated in the descriptor 'desc', a 16 bit value. If the file being created is a special file, the 'address' argument is used to specify the internal device number (a 16 bit value). The descriptor has the 'type' as the most significant byte and the 'permissions' as the least significant byte. Their definitions follow:

* types

```
 TPBLK => %00000010 ($02)  block type device
 TPCHR => %00000100 ($04)  character type device
 TPDIR => %00001000 ($08)  directory type file
```

* permissions

```
 FACUR => %00000001 ($01)  owner read permission
 FACUW => %00000010 ($02)  owner write permission
 FACUE => %00000100 ($04)  owner execute permission
 FACOR => %00001000 ($08)  others read permission
 FACOW => %00010000 ($10)  others write permission
 FACOE => %00100000 ($20)  others execute permission
 FXSET => %01000000 ($40)  set id bit for execute
```

DIAGNOSTICS

An error is issued if the file already exists or if the caller is not the system manager.

defacc

USAGE

```
  sys defacc,perm
```

DESCRIPTION

Set the default access permissions as specified by 'perm' (a 16 bit value). Normally, when a file is created, it is given the permissions specified in the 'create' system call. The value specified by 'create' is anded with the 1's compliment of a per task value known as the default permissions. This process will turn off or disable the permissions contained in the default permissions byte, no matter what the specified permissions are in the create call. The 'defacc' call is used to set the default permissions. All 'forks' and 'execs' pass on the existing default value. See 'chprm' for a list of the permission bits.

DIAGNOSTICS

No errors generated.

dup

USAGE

```
<file descriptor in D>
sys dup
<file descriptor in D>
```

DESCRIPTION

The file descriptor specified is duplicated. In other words, the file associated with the file descriptor is opened again and given another descriptor, which is returned. The new file is opened with the same mode as the original (e.g. if the original was open for 'read', so will the new one).

DIAGNOSTICS

An error if there are too many files opened, or if the file descriptor is invalid.

dups

USAGE

```
<file descriptor in D>
<specified descriptor in X>
sys dups
<file descriptor in D>
```

DESCRIPTION

This call is like 'dup' except the caller may specify the file descriptor of the duplicated open file. If the specified descriptor is already open, it is closed before the dup is done.

DIAGNOSTICS

An error if there are too many open files, or if the file descriptors are invalid.

exec

USAGE

```
  sys exec,fname,arglist
  ...
 fname fcc '....',0
  ...
 arglst fdb arg0,arg1,...,0
 arg0 fcc '....',0
 arg1 fcc '....',0
```

DESCRIPTION

Exec is the only way to execute a binary file. 'Fname' specifies the file to be executed. The calling task will be terminated and the new one started up. There is no return from a successful exec. A return indicates an error condition. All open files remain open through the exec. Interrupts which are being ignored will stay in that state, but those which are being caught are reset to their default state. When the file starts executing, the arguments are available as follows:

```
 ...  highest address in task space ($FE00) ...

         0
         ...
   arg0: <arg00>
         0
         argn
         ...
         arg0
   sp -> argcnt

 ...   low memory ...
```

The stack pointer is pointing at a 2 byte argument count. Above that is a list of pointers which point to the actual arguments which are at the highest part of memory. Two zero bytes are left at the very top of the task address space.

DIAGNOSTICS

An error will result (and a return to the caller of exec) if the file does not exist, it was not executable binary, there were too many arguments (approx. 3,000 bytes max), or the memory space was exceeded.

filtim

USAGE

```
  <timehi in X>
  <timelo in D>
  sys filtim,fname
```

DESCRIPTION

Filtim is used to set the 'last modified time' of a specified file. The file 'fname' will have its time set to the value contained in the D and X registers. Only the system manager may execute this call.

DIAGNOSTICS

An error is returned if the file does not exist, if the file is currently open by another task, or if the caller is not the system manager.

fork

USAGE

```
  sys fork
  <new task returns here>
  <old task here (pc+2), new task id in D>
```

DESCRIPTION

Fork is used to create a new task. The callers core image is copied to create the new task which inherits all open files and file pointers. The new task is identical to the original in every respect except that the old task returns 2 bytes past the system call and has the newly created task's id in the D register.

DIAGNOSTICS

An error results if too many tasks have been created or if the system task table is full.

gtid

USAGE

```
sys gtid
<task id in D>
```

DESCRIPTION

This call returns the running task's system id. This number may be used to generate unique file names.

DIAGNOSTICS

No errors are returned.

guid

USAGE

```
sys guid
<actual user id in D>
<effective user id in X>
```

DESCRIPTION

Guid returns both the actual user id and the effective user id. The actual id identifies the person who actually logged on the system while the effective id defines the current access permissions of the running task.

DIAGNOSTICS

No errors are possible.

ind

USAGE

```
  sys ind,label
```

DESCRIPTION

The ind system call is used where it is necessary to create system calls or their arguments on the fly (in the running program). The 'label' points to an address which contains the actual call and its arguments. The task resumes execution after the 'sys ind' and not after the labeled code. Another 'ind' or 'indx' call may not be called from ind.

DIAGNOSTICS

An error is issued if the value at the 'label' is not a valid system call, or if it is an indirect call.

USAGE

```
sys indx
```

DESCRIPTION

This call is similar to 'ind' but allows the system function code and arguments to be anywhere in memory, including the stack. Where 'ind' had a label pointing to the system call and parameters, this call requires X to point to the call and parameters. One application of 'indx' is to push the arguments and system call code on the stack, do an 'leax 0,s' to point to the call, then an 'indx'. Another indirection call may not be executed with this call.

DIAGNOSTICS

An error is reported if the system function is not a valid system call, or if it is another indirect call.

link

USAGE

```
  sys link,fname1,fname2
```

DESCRIPTION

This call is used to link 'fname1' and is given the name 'fname2'. After the link, reference to fname2 will access the contents of fname1. The files contents or attributes are not changed in any way.

DIAGNOSTICS

An error results if fname1 does not exist, if fname2 already exists, if the directory of fname2 is write protected, if fname1 is a directory, or if the file names are on different devices.

lock

USAGE

```
sys lock,flag
```

DESCRIPTION

Lock is used to lock a task in memory (keep it from being swapped). Only the system manager may execute this call. If 'flag' is non-zero, the task will be locked, if it is zero, the task will be unlocked.

DIAGNOSTICS

An error is issued if the caller is not the system manager.

lrec

USAGE

```
  <file descriptor in D>
  sys lrec,count
```

DESCRIPTION

Lrec is used to make an entry in the system's locked record table. All other entries in the table associated with the calling task and the specified file will be removed before making the new entry. The 'count' argument represents the number of bytes in the file (record size) to be locked from the current file position. If the specified record overlaps any part of another task's entry in the lock table for the same file, an error will result (ELOCK). Only regular files may be referenced (e.g. no devices, pipes, or directories). Closing a file will remove the lock table entry created as well as using the 'urec' system call. Note that the part of the file specified is not actually 'locked' from others use, but proper use of the lrec and urec calls will have the same efect.

DIAGNOSTICS

An error will result if there is no file for the specified descriptor, the file is not a regular file, the record is locked by another task, or the lock table is temporarily full.

USAGE

```
  sys mount,sname,fname,mode
```

DESCRIPTION

Mount is used to mount a special file on the file system. The file 'fname' should be a directory, and after the mount, any reference to 'fname' will reference the root directory of the special file (block device) 'sname'. The 'mode' is normally 0 but if non-zero, the device is mounted as 'read only' (i.e. writing not permitted).

DIAGNOSTICS

Errors are issued if 'sname' is not an appropriate file, if it is already mounted, if 'fname' does not exist, or if too many devices are currently mounted.

ofstat

USAGE

```
  <file descriptor in D>
  sys ofstat,buffer
```

DESCRIPTION

This call is used to get the status of an open file. The file is referenced by its file descriptor obtained when the file was opened or created. The status information is returned in the user space pointed at by 'buffer'. See the 'status' call for a description of the returned information.

DIAGNOSTICS

An error is returned if the file descriptor is not valid (i.e. file not open or descriptor out of range).

open

USAGE

```
sys open,fname,mode
<file descriptor in D>
```

DESCRIPTION

Open is used to open an existing file. The file is opened for reading if 'mode' is 0, for writing if 'mode' is 1, or both reading and writing if 'mode' is 2. The file name opened is 'fname'. Open returns a file descriptor which must be used for future file references.

DIAGNOSTICS

An error will be issued if the file does not exist, if the path directories cannot be searched, if too many files are open, or if the permissions do not grant the requested mode.

profile

USAGE

```
  sys profil,prpc,buffer,bsize,scale
```

DESCRIPTION

The profile call is used to set up a buffer and parameters to be used by the system to profile a running task. If profiling is enabled, each time a clock tick occurs (every tenth second) a word in the 'buffer' which corresponds to the current value of the program counter in the running task will be incremented. The 'prpc' value represents the lowest address in the running task to be profiled. The address of the profile buffer is given by 'buffer' and its size by 'bsize'. The buffer size also determines the highest address in the running task to be profiled since pc addresses too large to be mapped into the buffer are ignored. The 'scale' value is used to scale the task program counter and must be a power of 2 (maximum size is 128). Profiling may be disabled by setting scale to 0 or 1. While profiling a task, the following happens every time the task is running and a clock interrupt occurs. The profile value 'prpc' is subtracted from the tasks current program counter and the result is divided by the scale factor. This value is then multiplied by 2 to form an offset into the 'buffer'. If this offset is less than 'bsize', the 16 bit word residing at 'buffer'+'offset' is incremented by one.

DIAGNOSTICS

No errors are issued.

USAGE

```
<file descriptor in D>
sys read,buffer,count
<bytes read in D>
```

DESCRIPTION

This call is used to read the file represented by the file descriptor specified. The space pointed to by 'buffer' in the user's space is filled with data from the file. A maximum of 'count' bytes will be read. All bytes requested will not necessarily be returned. If the file is a terminal, one line will be returned at most. If the returned byte count is zero, and no error is reported, the end of file has been reached.

DIAGNOSTICS

Errors are issued if there was a physical i/o error, bad file descriptor, or bad count specified.

sacct

USAGE

```
sys sacct,fname
```

DESCRIPTION

This call is used to enable or disable system accounting. The argument 'fname' is a pointer to a null terminated string representing the path name of the file to be used to collect the accounting data. The file must already exist or an error will result. If 'fname' is a null pointer ($0000), accounting will be turned off. When system accounting is enabled, the system will write a record to the specified file each time a task terminates. Each record will contain the following information:

```
* sacct file record

 acuid     rmb  2   user id number
 acstrt    rmb  4   starting time of task
 acend     rmb  4   ending time of task
 acsyst    rmb  3   system time used by task
 acusrt    rmb  3   user time used by task
 acstat    rmb  2   task termination status
 actty     rmb  1   task terminal number
 acmem     rmb  1   maximum memory used (4K blocks)
 acblks    rmb  2   io units (measure of blocks read or written)
 acspar    rmb  2   spare
 acname    rmb  8   command name
```

Each new record written to the file is appended to the end of the file. Only the system manager may execute this call.

DIAGNOSTICS

An error is returned if the caller is not the system manager, if the file does not exist, or if an attempt is made to enable accounting when already active.

seek

USAGE

```
<file descriptor in D>
sys seek,positionhi,positionlo,type
<hi position in X>
<lo position in D>
```

DESCRIPTION

The file represented by the file descriptor will have its read/write pointer positioned to the specified file location. The arguments 'positionhi' and 'positionlo' represent a four byte, signed offset. The starting point for this offset is determined as follows by the 'type' argument:

type	starting position
0	Position from the beginning of the file
1	Position from the current position
2	Position from the end of the file

The returned value is the resulting position of the file. If a 'seek' is performed past the end of the file when writing, a gap in the file will be created (no actual device space will be allocated). This gap will be read as zeros. To determine the current position in the file, use 'sys seek,0,0,1'.

DESCRIPTION

An error is returned if a bad file descriptor is used or attempting a seek on a pipe.

setpr

USAGE

```
  <priority in D>
  sys setpr
```

DESCRIPTION

Setpr is used to set the priority bias used by the system scheduler. The value specified is subtracted from the normal user priority, so the effect is that of lowering the task's priority. Only the system manager may specify negative arguments (which will increase the task's priority). The priority bias specified should be in the range of 25 to -25.

DIAGNOSTICS

No errors are issued.

spint

USAGE

```
<task number in D>
sys spint,interrupt
```

DESCRIPTION

This call is used to send a program interrupt to a task. The task is specified by its task number and the receiving task must have the same effective user id unless the caller is the system manager. The 'interrupt' argument specifies which interrupt to send. See 'cpint' for a list of interrupts. If the task number specified is zero, the interrupt will be sent to all tasks associated with the caller's control terminal. If the task number is -1, and if the caller is the system manager, the interrupt is sent to all tasks in the system with the exception of tasks 0 and 1 (the scheduler and the initializer).

DIAGNOSTICS

An error is issued if the task specified does not exist or if the effective user id's do not match.

stack

USAGE

```
<address in X>
sys stack
```

DESCRIPTION

The system will extend the user's stack memory to include the address specified. If the address is higher than what is currently allocated, all lower memory will be released to the system. A task initially starts with stack space between 100 and 3000 bytes depending on the number of arguments passed from exec.

DIAGNOSTICS

An error results if the request for memory overflows into the data segment.

status

USAGE

```
sys status,fname,buffer
```

DESCRIPTION

The file 'fname' has its status read and returned to the user in the space specified by 'buffer'. The data returned by this call (as well as 'ofstat') has the following format:

```
  * buffer begin *

 st_dev    rmb   2     device number
 st_fdn    rmb   2     fdn number
 st_mod    rmb   1     file modes - see below -
 st_prm    rmb   1     permission bits - see below -
 st_cnt    rmb   1     link count
 st-own    rmb   2     file owner's user id
 st_siz    rmb   4     file size in bytes
 st_mtm    rmb   4     last time file was modified
 st_spr    rmb   4     future use only

* mode codes

 FSBLK => %00000010  ($2)  block device
 FSCHR => %00000100  ($4)  character device
 FSDIR => %00001000  ($8)  directory

* permissions

 FACUR => %00000001 ($01)  owner read permission
 FACUW => %00000010 ($02)  owner write permission
 FACUE => %00000100 ($04)  owner execute permission
 FACOR => %00001000 ($08)  others read permission
 FACOW => %00010000 ($10)  others write permission
 FACOE => %00100000 ($20)  others execute permission
 FXSET => %01000000 ($40)  set id bit for execute
```

DIAGNOSTICS

An error is issued if the file does not exist or the directory path cannot be searched.

stime

USAGE

```
<timehi in X>
<timelo in D>
sys stime
```

DESCRIPTION

The system time and date is set. The time is measured in seconds from 0000 UTC January 1, 1980. Only the system manager may execute this call.

DIAGNOSTICS

An error is reported if the caller is not the system manager.

USAGE

```
  sys stop
```

DESCRIPTION

Stop is used to halt a task until a program interrupt is received from 'spint' or 'alarm'. When stop returns, it will always have an error (EINTR).

DIAGNOSTICS

See above.

suid

USAGE

```
<user id in D>
sys suid
```

DESCRIPTION

This call is used to set the effective and actual user id. This call may only be executed if the actual user id matches the id in the argument, or if the caller is the system manager.

DIAGNOSTICS

An error if the caller is not the system manager or if the actual user id does not match.

term

USAGE

```
  <status in D>
  sys term
```

DESCRIPTION

Term is used to terminate a task. The status specified is made available to the parent task and usually zero if there were no errors in the terminating task. A non-zero status should indicate some error condition. This system call never returns to the caller.

DIAGNOSTICS

No errors reported.

time

USAGE

```
  sys time,tbuf
```

DESCRIPTION

The 'time' call is used to get the systems idea of the current time. Internally, the time is kept as a four byte number, representing the number of seconds which have elapsed since 0000 January 1, 1980 UTC. The time information is placed at the address specified by 'tbuf' and has the following format:

```
* system time information

 tm_sec  rmb  4   time in seconds
 tm_tik  rmb  1   ticks in current second (tenths)
 tm_zon  rmb  2   time zone
 tm_dst  rmb  1   daylight savings flag
```

The 'tm_tik' value may be used for finer measurements. The timezone word is the number of minutes of time westward from Greenwich (eastward would be a negative number). If 'tm_dst' is non-zero, it implies that the local time zone should be altered for Daylight Savings during the appropriate part of the year.

DIAGNOSTICS

No errors are issued.

USAGE

```
  sys trap,address
  <previous trap address in D>
```

DESCRIPTION

The trap system call is used to set the swi2 vector in the 6809. This call is very machine and configuration dependent and may not be supported on all systems. Each task has its own vector. The value returned by trap is the previous value of the vector. When a swi2 instruction is executed, control will transfer to the address set in trap vector. An RTI instruction will resume execution after the swi2.

DIAGNOSTICS

No errors are issued.

ttime

USAGE

```
sys ttime,buffer
```

DESCRIPTION

This call is used to obtain the accounting time information about a task. All times are represented in tenths of seconds. The information is returned to the user at 'buffer' and has the following format:

```
* ttime buffer

 ti_usr  rmb  3   task's user time
 ti_sys  rmb  3   task's system time
 ti_chu  rmb  4   children's user time
 ti_chs  rmb  4   children's system time
```

The child times shown are the totals of all children tasks spawned by this task and its children.

DIAGNOSTICS

No errors are issued.

ttyget

USAGE

```
  <file descriptor in D>
  sys ttyget,ttbuf
```

DESCRIPTION

This call is used to return information about a terminal. The information returned is but in the 6 byte buffer pointed at by 'ttbuf'. The data has the following format:

```
   * ttbuf *

 tt_flg  rmb  1  flags byte - see below -
 tt_dly  rmb  1  delay byte - see below -
 tt_cnc  rmb  1  line cancel char (default is ©X)
 tt_bks  rmb  1  backspace character (default is ©H)
 tt_spd  rmb  1  terminal speed - see below -
 tt_spr  rmb  1  reserved for future use

* flags

 RAW   => %00000001  ($01)  raw i/o mode
 ECHO  => %00000010  ($02)  echo input characters
 XTABS => %00000100  ($04)  expand tabs on output
 LCASE => %00001000  ($08)  map upper->lower on input and vice versa
 CRMOD => %00010000  ($10)  output cr and lf for cr
 BSECH => %00100000  ($20)  echo backspace echo char
 SCHR  => %01000000  ($40)  single character input mode
 CNTRL => %10000000  ($80)  ignore control characters mode

* delays

 DELNL => %00000011  ($03)  new line delay
 DELCR => %00001100  ($0C)  c.r. delay
 DELTB => %00010000  ($10)  tab delay
 DELVT => %00100000  ($20)  vertical tab delay
 DELFF => %00100000  ($20)  form feed (same as DELVT)

* speeds

 INCHR => %10000000  ($80)  input ready to be consumed
```

DIAGNOSTICS

An error is returned if specified file is not a character type device.

ttynum

USAGE

```
sys ttynum
<terminal number in D>
```

DESCRIPTION

This call is used to return the number of the calling task's terminal. As an example, "tty02" would return $0002 in the D register.

DIAGNOSTICS

No errors are issued.

ttyset

USAGE

```
<file descriptor in D>
sys ttyset,ttbuf
```

DESCRIPTION

This call sets the terminal information described in 'ttyget'. The data in 'ttbuf' is exactly as described in 'ttyget'.

DIAGNOSTICS

An error is issued if the file specified is not a character type device.

unlink

USAGE

```
sys unlink,fname
```

DESCRIPTION

Unlink is used to remove the 'fname' entry from a directory. If this is the last link to the file, the file will be deleted and its device space will be freed. If the file is open, the file will not be destroyed until the file is closed.

DIAGNOSTICS

An error is issued if the file does not exist, if the directory cannot be written, or if the directory path cannot be searched.

USAGE

```
  sys unmnt,sname
```

DESCRIPTION

The special file 'sname' is unmounted from the system. The file which was associated with the special file will revert to its ordinary interpretation (see mount).

DIAGNOSTICS

An error is issued if the file system specified is busy or is not mounted.

update

USAGE

```
sys update
```

DESCRIPTION

Update is used to update all information on the disks. Any data which is in memory destined for disk will be written out at this time.

DIAGNOSTICS

No errors are reported.

USAGE

```
  <file descriptor in D>
  sys urec
```

DESCRIPTION

Urec is used to remove an entry in the system's lock table which was previously installed by 'lrec'. All entries associated with the calling task and specified file are removed.

DIAGNOSTICS

An error is issued if the file descriptor specified is bad.

wait

USAGE

```
sys wait
<task id in D>
<term status in X>
```

DESCRIPTION

This call is used to wait for a program interrupt or the termination of a child task. If several children tasks have been spawned it is necessary to execute a 'wait' for each one. The task id of the terminated task is returned as well as its termination status. The low byte of this status is the value passed by the 'term' system call. A non-zero value here usually represents some sort of error condition. The high byte of the status is zero for normal termination. If non-zero, this byte will contain the interrupt number which caused it to terminate. If the most significant bit of the status is set, a core dump was produced as a result of termination. Consult 'cpint' for a list of interrupt numbers.

DIAGNOSTICS

An error is issued if there are no children tasks.

write

USAGE

```
<file descriptor in D>
sys write,buffer,count
<byte count written in D>
```

DESCRIPTION

Write is used to write data to a file. The file specified by the file descriptor has 'count' bytes written from location 'buffer' to it. If the returned byte count does not equal the requested 'count', it should be considered an error. Writes which are multiples of 512 bytes and begin on 512 byte address boundaries are the most efficient.

DIAGNOSTICS

An error is issued if the file descriptor is invalid or if a physical i/o error resulted.