68xxx UniFLEX®

Introduction to UniFLEX® System Calls

MANUAL REVISION HISTORY

Revision	Date	Change
A	05/84	Original release
B	03/86	Revised and updated documentation for all system calls. Added new calls.
C	09/86	Manual Update for Version 2.0 of 68xxx UniFLEX. New system calls: stack_limit Revised software: cpint, spint Revised text only: control_pty, create_contiguous, crpipe, crtsd, defacc, fcntl, FPU_exception, link, mount, msg_attach, msg_receive, msg_send, msg_status, ofstat, sacct, setpr, status, time, ttime, ttyget, ttyset, yield_CPU

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

All programs interface with the 68xxx UniFLEX Operating System (UniFLEX Registered in U.S. Patent and Trademark Office) through system calls. The operating system supports a variety of calls, which allow the manipulation of files, the control of tasks, and other system functions. The assemblers have a built-in instruction, "sys", to make system calls to the operating system. The syntax for this system call follows:

```
sys <function_code> [<param_list>]
```

where <function_code> is a 16-bit code for the desired system call (the file "/lib/sysdef" defines the correspondence between function codes and system calls). The number and type of parameters in the list may range from 0 to 4 inclusive. The parameters are always 32-bit values.

Many of the system calls expect certain values or parameters to be in one or more of the 68xxx CPU's registers. The documentation for a system call shows any such items in the section entitled ASSEMBLY LANGUAGE SYNTAX under the subheading "Expected". In such cases it is the programmer's responsibility to load the proper values before invoking the system call.

When the "sys" instruction completes execution, control generally passes to the next instruction in the program. In some cases the system call must return one or more values to the calling program. Generally, a system call places such a value in a register although in some cases it places it in a location specified in <param_list>. The documentation for a system call shows any values returned in a register in the section entitled ASSEMBLY LANGUAGE SYNTAX under the subheading "Returns".

The operating system preserves all registers throughout the execution of a successful system call unless the system call returns a value in a register. If the system call fails, the call always returns the error number of the error that caused the failure in the D0 register.

The operating system also preserves all condition codes throughout a system call with the exception of the error (or carry) bit. If an error occurs, the system call sets the error bit; otherwise, the bit is cleared. The assemblers support the mnemonics "bes" (branch if error set) and "bec" (branch if error clear), which are synonymous with "bcs" and "bcc".

Several system calls, such as "read" and "ttime" return data to a buffer whose address is specified when the call is invoked. In such a case the buffer must be in either the data segment or the stack. It may not be located in the text segment.

Several files containing definitions used by the operating system reside in the directory "/lib". The user can include any of these definitions in a program by using the appropriate file as a library. A description of each of these files follows:

sysdef	Defines the correspondence between the names and numbers of UniFLEX system calls.
syserrors	Defines the correspondence between the names and numbers of UniFLEX errors. Also contains a brief definition of the most general cause of each error.
sysfcntl	Defines the correspondence between the names and numbers of the subfunctions used by the "fcntl" system call. Also defines the constants used by this system call.
sysints	Defines the correspondence between the names and numbers of UniFLEX interrupts. Also contains a brief definition of each interrupt.
sysmessages	Defines the structure of the buffer returned by the "msg_status" system call.
syspty	Defines the correspondence between the names and the numbers of the subfunctions used by the "control_pty" system call. Also defines the constants used by this system call.
sysrump	Defines the correspondence between the names and the numbers of the subfunctions used by the "rump" system call.
sysstat	Defines the structure of the buffer returned by the "ofstat" and "status" system calls. Also defines the file-permission flags.
systime	Defines the structures of the buffers returned by the "time" and "ttime" system calls.
systty	Defines the structure of the buffer returned by the "ttyget" and "ttyset" system calls. Also defines the constants used by these system calls.
sys68881	Defines the structure of the buffer used by the "FPU_exception" system call.

Syntax Conventions

The following conventions are used in syntax statements throughout this manual.

1. Items that are not enclosed in angle brackets, '<' and '>', or square brackets, '[' and ']', are "keywords" and should be typed as shown.

2. Angle brackets, '<' and '>', enclose descriptions which the user must replace with a specific argument. Expressions enclosed only in angle brackets are essential parts of the syntax. For example, in the system call

   sys ind,<call_address>

   the address of the system call to execute must replace the term <call_address>.

3. Square brackets, '[' and ']', indicate optional items. These items may be omitted if their effect is not desired.

4. The underscore character, '_', is used to link separate words that describe one term, such as "call" and "address".

5. Characters other than spaces that are not enclosed in angle brackets or square brackets must appear in the command line as they appear in the syntax statement.

alarm	Send an alarm interrupt to the current task after the specified interval.
break	Change the size of the data segment associated with the task.
cdata	Change the amount of physically contiguous memory associated with the task.
chacc	Check the permissions on the specified file.
chdir	Make the specified directory the user's working directory.
chown	Change the owner of a file.
chprm	Change the access permissions of the specified file.
close	Close an open file.
control_pty	Adjust or report the modes of operation of a pseudoterminal.
cpint	Inform the operating system how to behave when the current task receives one kind of interrupt.
create	Create a new file or truncate an existing file to a length of 0.
create_contiguous	Create a new contiguous file.
create_pty	Open an unused pseudoterminal channel.
crpipe	Create a pipe.
crtsd	Create a special file (a device) or a new directory.
defacc	Set the default access permissions for all files created by this task.
dup	Duplicate the specified file descriptor.
dups	Duplicate a file descriptor onto the specified file descriptor.
exec	Execute a program.
exece	Execute a program.
fcntl	Change or query the behavior of a file.
filtim	Change the time of last modification of the specified file.
fork	Create a new task.
FPU_exception	Return or update information about an exception generated by the MC68881 floating-point coprocessor.
FPU_resume	Resume execution of the MC68881 instruction that generated an exception.
gpid	Get the task ID of the parent of the current task.
gtid	Get the task ID of the current task.
guid	Return the user ID and the effective user ID of the person executing the current task.

(continued)

ind	Execute the system call located at the specified address.
indx	Execute the system call located at the specified address.
link	Create a link to a file.
lock	Lock a task in memory or unlock a locked task.
lrec	Add an entry to the operating system's lock table.
make_realtime	Make a non-real-time task a real-time task and set its relative priority, or make a real-time task a non-real-time task.
memman	Perform a memory-management operation.
mount	Insert the medium in the specified block device at the node of the directory tree specified by <dir_name>.
msg_attach	Attach a task to a message exchange.
msg_detach	Detach a task from a message exchange.
msg_receive	Receive a message from another task via a message exchange.
msg_send	Send a message to another task via a message exchange.
msg_status	Obtain information about the status of a message exchange.
ofstat	Get the status of an open file.
open	Open an existing file.
phys	Obtain or release access to a section of system memory.
profile	Start or stop monitoring the current task.
read	Read data from an open file.
rump	Create, destroy, access, or relinquish access to a named resource.
sacct	Enable or disable system accounting procedures.
seek	Change the current file position of an open file.
setpr	Change the priority bias of the current task.
set_high_address_mask	Load the specified value into the register for the hardware address mask.
spint	Send an interrupt to a task.
stack	Extend the stack space of the current task.
stack_limit	Specify a limit to the task's stack segment.
status	Get the status of a file.
stime	Set the system date and time.
stop	Suspend the current task.
suid	Set both the user ID and the effective user ID.
term	Terminate the current task.
time	Get the system time and other related parameters.
truncate	Set the size of an open file.
ttime	Get the information on the use of the CPU by the current task and its child tasks.
ttyget	Get information on the configuration of a terminal.

(continued)

ttynum	Get the terminal number of the task's controlling terminal.
ttyset	Set the configuration of a terminal.
unlink	Remove a link to a file.
unmnt	Unmount the medium in a device.
update	Update all disks on the system.
urec	Remove an entry from the operating system's lock table.
vfork	Create a new task.
wait	Suspend the task until a child task terminates.
write	Write data to an open file.
yield_CPU	Yield the CPU to another task of equal priority.

```
Expected: <seconds> in D0
Syntax:   sys alarm
Returned: <previous_seconds> in D0

Syntax:   sys break,<high_address>

Syntax:   sys cdata,<high_address>

Syntax:   sys chacc,<file_name>,<perm_mask>

Syntax:   sys chdir,<dir_name>

Syntax:   sys chown,<file_name>,<owner_ID>

Syntax:   sys chprm,<file_name>,<perm_mask>

Expected: <file_des> in D0
Syntax:   sys close

Expected: <file_des> in D0
Syntax:   sys control_pty,<function_code>,<mode_flag>
Returns:  <state_flag> in D0

Syntax:   sys cpint,<interrupt>,<address>
Returns:  <old_address> in D0

Syntax:   sys create,<file_name>,<perm_mask>
Returns:  <file_des> in D0

Syntax:   sys create_contiguous,<file_name>,<perm_mask>,<file_size>,
                                <zero_flag>
Returns:  <file_des> in D0

Expected: <slave_file_des> in D0
          <master_file_des> in A0
Syntax:   sys create_pty

Syntax:   sys crpipe
Returns:  <read_file_des> in D0
          <write_file_des> in A0

Syntax:   sys crtsd,<file_name>,<descriptor>,<dev_num>

Syntax:   sys defacc,<perm_mask>
Returns:  <previous_mask> in D0
```

```
Expected: <file_des> in D0
Syntax:   sys dup
Returns:  <new_file_des> in D0

Expected: <file_des> in D0
          <requested_file_des> in A0
Syntax:   sys dups
Returns:  <requested_file_des> in D0

Syntax:   sys exec,<file_name>,<arg_list>

Syntax:   sys exece,<file_name>,<arg_list>,<env_list>

Expected: <file_des_1> in D0
Syntax:   sys fcntl,<function_code>
Returns:  <access_flag> in D0
  or
          <file_des_2> in D0

Expected: <time> in D0
Syntax:   sys filtim,<file_name>

Syntax:   sys fork
Returns:  To parent task: <child_task's_ID> in D0
          To child task:  0 in D0

Syntax:   sys FPU_exception,<function_code>,<buf_add>

Syntax:   sys FPU_resume

Syntax:   sys gpid
Returns:  <parent_task_ID> in D0

Syntax:   sys gtid
Returns:  <task_ID> in D0

Syntax:   sys guid
Returns:  <user_ID> in D0
          <effective_user_ID> in A0

Syntax:   sys ind,<call_address>

Expected: <call_address> in A0
Syntax:   sys indx

Syntax:   sys link,<file_name_1>,<file_name_2>

Syntax:   lock,<function_code>

Expected: <file_des> in D0
Syntax:   sys lrec,<count>
```

```
Expected: <relative_priority> in D0
Syntax:   sys make_realtime

Syntax:   sys memman,<function_code>,<start_address>,<end_address>

Syntax:   sys mount,<dev_name>,<dir_name>,<mode>

Syntax:   sys msg_attach,<exchange_name>,<mode>
Returns:  <exchange_ID> in D0

Expected: <exchange_ID> in D0
Syntax:   sys msg_detach

Expected: <exchange_ID> in D0
Syntax:   sys msg_receive,<buf_add>,<mode>

Expected: <exchange_ID> in D0
Syntax:   sys msg_send,<buf_add>,<mode>

Expected: <exchange_ID> in D0
Syntax:   sys msg_status,<buf_add>

Expected: <file_des> in D0
Syntax:   sys ofstat,<buf_add>

Syntax:   sys open,<file_name>,<mode>
Returns:  <file_des> in D0

Syntax:   sys phys,[-]<code>
Returns:  <log_base_add> in D0  (only when obtaining access to memory)

Syntax:   sys profile,<start_add>,<buf_add>,<size>,<scale>

Expected: <file_des> in D0
Syntax:   sys read,<buf_add>,<count>
Returns:  <bytes_read>

Expected: <function_code> in D0
          <resource_name> in A0
Syntax:   sys rump

Syntax:   sys sacct,<file_name>

Expected: <file_des> in D0
Syntax:   sys seek,<count>,<pt_of_origin>
Returns:  <new_position>

Expected: <priority_bias> in D0
Syntax:   sys setpr
```


(continued)

```
Expected: <address_mask> in D0
Syntax:   sys set_high_address_mask


Expected: <task_ID> in D0
Syntax:   sys spint,<interrupt>

Expected: <address> in D0
Syntax:   sys stack

Expected: <address> in A0
Syntax:   sys stack_limit
Returns:  <previous_limit> in D0

Syntax:   sys status,<file_name>,<buf_add>

Expected: <time> in D0
Syntax:   sys stime

Syntax:   sys stop

Expected: <user_ID> in D0
Syntax:   sys suid

Expected: <term_status> in D0
Syntax:   sys term

Syntax:   sys time,<buf_add>

Expected: <file_des> in D0
Syntax:   sys truncate

Syntax:   sys ttime,<buf_add>

Expected: <file_des> in D0
Syntax:   sys ttyget,<buf_add>

Syntax:   sys ttynum
Returns:  <tty_num> in D0

Expected: <file_des> in D0
Syntax:   sys ttyset,<buf_add>

Syntax:   sys unlink,<file_name>

Syntax:   sys unmnt,<dev_name>

Syntax:   sys update

Expected: <file_des> in D0
Syntax:   sys urec
```

```
Syntax:   sys vfork
Returns:  To parent task: <child_task's_ID> in D0
          To child task:  0 in D0

Syntax:   sys wait
Returns:  <task_ID> in D0
          <term_status> in A0

Expected: <file_des> in D0
Syntax:   sys write,<buf_add>,<count>
Returns:  <bytes_written>

Syntax:   sys yield_CPU
```

alarm

Send an alarm interrupt to the current task after the specified interval.

ASSEMBLY LANGUAGE SYNTAX

Expected

<seconds> in D0

Syntax Statement

sys alarm

Returns

<previous_seconds> in D0

Arguments

<seconds>	The number of seconds to wait before sending the alarm interrupt. A value of 0 cancels any existing request for an alarm interrupt and does not generate a new one.
<previous_seconds>	The number of seconds remaining from the previous request for an alarm interrupt or 0 if the user made no previous request.

DESCRIPTION

The "alarm" system call sets the task's alarm clock to send an alarm interrupt (SIGALARM) after the interval specified in the D0 register has elapsed. If the alarm clock is already set, the call overrides the previous setting. Unless the program catches or ignores it, the interrupt terminates the task. The system call returns control to the caller immediately after execution.

NOTES

- The time that elapses before the system sends the alarm interrupt may be slightly less than the requested time because the system ticks occur at one-second intervals.

. The time that elapses before the system sends an alarm interrupt to a task that invoked the call with a value of 1 for <seconds> may be more than 1 second if the time until the next 1-second interval in the system is small.

SEE ALSO

cpint
stop

break

Change the size of the data segment associated with the task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys break,<high_address>
```

Arguments

<high_address>	The highest address to be used by the task for data space.

DESCRIPTION

The "break" system call sets the end-of-segment address for the data segment of the task. If <high_address> is higher than the data segment's current end-of-segment address, "break" allocates memory to the segment; if it is lower, "break" relinquishes memory to the system.

ERRORS REPORTED

EDTOF
Not enough memory is available to honor the request.

SEE ALSO

cdata

cdata

Change the amount of physically contiguous memory associated with the task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys cdata,<high_address>
```

Arguments

<high_address>	The highest address to be used by the task for data space.

DESCRIPTION

The "cdata" system call sets the end-of-segment address for the data segment of the task. The system allocates physically contiguous memory to the task and locks the contiguous pages in memory so that they will remain contiguous. If <high_address> is higher than the data segment's current end-of-segment address, "cdata" allocates memory to the segment; if it is lower than the data segment's current address but higher than the lowest address in the data segment, "cdata" relinquishes memory to the system; if it is lower than the lowest address in the data segment, "cdata" neither allocates nor releases memory.

ERRORS REPORTED

EDTOF
: Not enough memory is available to honor the request.

SEE ALSO

break

chacc

Check the permissions on the specified file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys chacc,<file_name>,<perm_mask>
```

Arguments

<file_name> The address of the null-terminated name of the file to check.

<perm_mask> An 8-bit mask specifying which permissions to check for. If the value of <perm_mask> is 0, the system call checks for the existence of the specified file and for execute permission on all directories in the path leading to the file. The following table shows the correspondence between other values of <perm_mask> and the permissions to check. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of these values is valid.

<perm_mask>	Permission to Check
0001	Read (FACUR or FACOR)
0010	Write (FACUW or FACOW)
0100	Execute (FACUE or FACOE)

DESCRIPTION

The "chacc" system call checks to see whether or not the specified permissions are set for the user on <file_name>.

ERRORS REPORTED

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The permissions set on the specified file do not grant the requested type of access.

SEE ALSO

chprm

chdir

Make the specified directory the user's working directory.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys chdir,<dir_name>
```

Arguments

<dir_name>	The address of the null-terminated name of the directory to be the working directory.

DESCRIPTION

The "chdir" system call changes the working directory to that specified by <dir_name>.

ERRORS REPORTED

EMSDR
  The path to <file_name> cannot be followed.

ENDR
  A part of the path to <file_name> is not a directory.

chown

Change the owner of a file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys chown,<file_name>,<owner_ID>
```

Arguments

<file_name>	The address of the null-terminated name of the file whose owner is to change.
<owner_ID>	The user identification number (ID) of the new owner of the specified file. It need not be an ID found in the password file. The maximum permissible value is hexadecimal FFFF.

DESCRIPTION

The "chown" system call changes the owner of the specified file to the user specified by <owner_ID>. Only the system manager may invoke this system call.

ERRORS REPORTED

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The current effective user is not the system manager.

SEE ALSO

chprm
ofstat
stat

chprm

Change the access permissions of the specified file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys chprm,<file_name>,<perm_mask>
```

Arguments

<file_name> The address of the null-terminated name of the file whose permissions are to change.

<perm_mask> An 8-bit mask specifying the new permissions for the file. The following table shows the permission that is assigned to the file as each bit is set. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

<perm_mask>	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users. When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

DESCRIPTION

The "chprm" system call changes the set of access permissions associated with the specified file to those described in <perm_mask>. Only the system manager or the owner of the specified file may execute this system call.

ERRORS REPORTED

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The current effective user is neither the system manager nor the owner of the specified file.

SEE ALSO

chown
ofstat
stat

close

Close an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

    <file_des> in D0

Syntax Statement

    sys close

Arguments

    <file_des> The file descriptor of the file to close.

DESCRIPTION

The "close" system call closes the specified open file. The operating system automatically closes all files used by a task when the task terminates, but it is good practice to explicitly close a file as soon as the task is finished with it. As it closes a file, "close" checks its link count. If the link count is 0, "close" deletes the file from the system.

ERRORS REPORTED

EBADF
  The file descriptor does not reference an open file.

EBARG
  An argument to the system call is invalid.

SEE ALSO

create
create_pty
crpipe
dup
dups
open

control_pty

Adjust or report the modes of operation of a pseudoterminal.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys control_pty,<function_code>,<mode_flag>

Returns

<state_flag> in D0

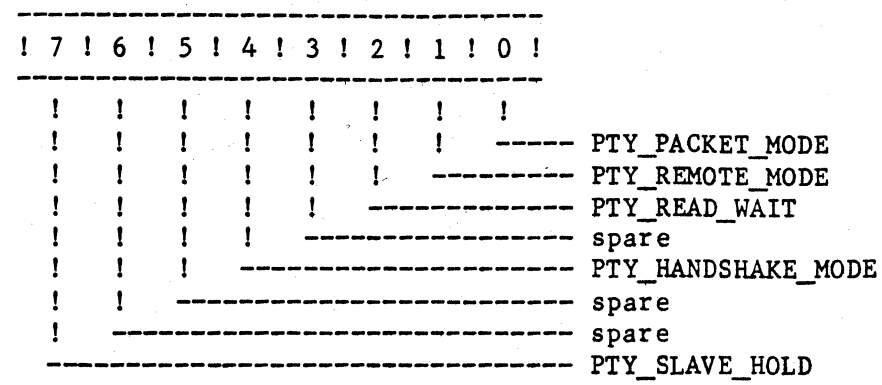
Arguments

<file_des>	The file descriptor of the master task of the pseudoterminal to control.
<function_code>	The subfunction to perform.
<mode_flag>	A flag used with PTY_SET_MODE to describe the desired modes of behavior of the pseudoterminal. All other subfunctions ignore this flag, but because the system call expects this argument, a value of 0 should be used in the syntax statement.
<state_flag>	A flag describing the current state of the specified pseudoterminal.

DESCRIPTION

The operating system creates a pseudoterminal in a particular state which includes both modes of behavior which the user can directly alter and conditions reflecting the current usage of the pseudoterminal, which the user cannot directly alter. The "control_pty" system call adjusts or reports the state of the specified pseudoterminal.

The modes of behavior are defined in the low-order byte of <state_flag>, which is always returned to D0 by "control_pty". When the operating system creates a pseudoterminal, none of these bits is set--that is, none of the modes of behavior is in effect. The structure of this byte, which is defined in the file "/lib/syspty", is as follows:



By default, a "read" call from the master task returns only the data written by the slave task. If PTY_PACKET_MODE is in effect (bit 0 is set), a "read" system call returns two bytes of information in addition to any data read. If data are available, these additional bytes are both null. Otherwise, they contain information describing the state of the pseudoterminal--that is, <state_flag>.

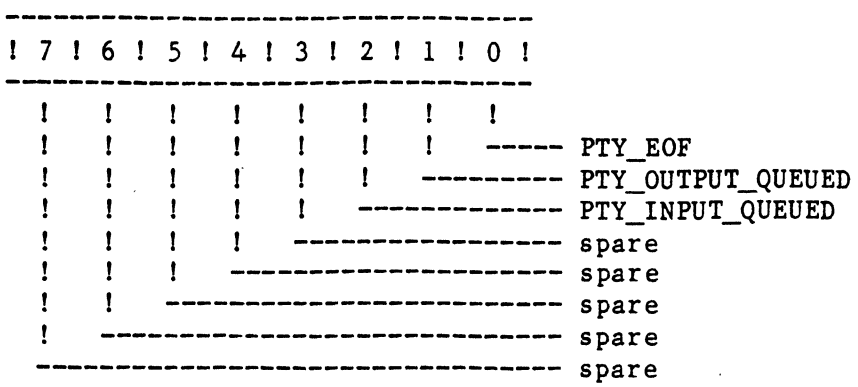
By default, characters written from the master task to the pseudoterminal are processed normally, not as if the channel to the master task were in raw mode. If PTY_REMOTE_MODE is in effect, the pseudoterminal treats all input it receives from the master task as if the channel to the master task were in raw mode.

By default, the operating system performs a "read" system call from the master task whether or not data are available and completes a "write" system call before a slave has consumed the data. If PTY_READ_WAIT mode is in effect, the system puts a "read" system call from the master task to sleep until data are available. Similarly, if PTY_HANDSHAKE_MODE is in effect, it puts a "write" system call from the master task to sleep until a slave has consumed the data.

Finally, by default the system does nothing to prevent any slave task from writing to the pseudoterminal. If, however, PTY_SLAVE_HOLD mode is in effect, the system prohibits all slave tasks from writing to the pseudoterminal.

The high-order byte of <state_flag> reflects conditions that the user cannot alter with the "control_pty" command. The structure of this byte, which is defined in the file "/lib/syspty", is as follows:

(continued)



When bit 0 (PTY_EOF) is set, all accesses to the pseudoterminal from slave tasks are closed. When bit 1 (PTY_OUTPUT_QUEUED) is set, a slave task has written data to the pseudoterminal, but the master task has not yet consumed it. When bit 2 (PTY_INPUT_QUEUED) is set, the master has written data to the pseudoterminal, but a slave task has not yet consumed it.

The "control_pty" system call supports six subfunctions, which are defined in the file "/lib/syspty" as follows:

Code	Subfunction
0	PTY_INQUIRY
1	PTY_SET_MODE
2	not used
3	PTY_FLUSH_READ
4	PTY_FLUSH_WRITE
5	PTY_STOP_OUTPUT
6	PTY_START_OUTPUT

A description of each of these subfunctions follows:

PTY_INQUIRY	Report the current state of the pseudoterminal, but alter nothing.
PTY_SET_MODE	Rewrite the low-order byte of <state_flag> from the value of <mode_flag> specified by the user. This subfunction allows the user to alter all modes of behavior simultaneously.
PTY_FLUSH_READ	Delete any data written by the master task that has not yet been consumed by a slave task.
PTY_FLUSH_WRITE	Delete any data written by a slave task that has not yet been consumed by the master task.

PTY_STOP_OUTPUT	Prevent all slave tasks from writing any more data to the master task (enable PTY_SLAVE_HOLD mode).
PTY_START_OUTPUT	Allow all slave tasks to write data to the master task (disable PTY_SLAVE_HOLD mode).

NOTES

. Pseudoterminals are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBARG
An argument to the function is invalid.

EIO
The file descriptor specified corresponds to access in slave mode; it must correspond to access in master mode.

SEE ALSO

create_pty

cpint

Inform the operating system how to behave when the current task receives one kind of interrupt.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys cpint,<interrupt>,<address>
```

Returns

<old_address> in D0

Arguments

<interrupt>	The kind of interrupt to be handled by this system call.
<address>	The interrupt-handling address for <interrupt>--that is, the address to which control should pass whenever the current task receives the specified kind of interrupt. A value of 0 tells the operating system to take the default action for the interrupt (usually termination of the task). A value of 1 or any other odd address tells the operating system to ignore the interrupt.
<old_address>	The previous interrupt-handling address for the specified kind of interrupt.

DESCRIPTION

The "cpint" system call tells the operating system how to behave when the current task receives the specified interrupt. When the interrupt-handling address is an even, nonzero address, the operating system passes control to that address when the current task receives the specified interrupt. After processing the interrupt as specified at the interrupt-handling address, the operating system resets itself so as to take the default action the next time the current task receives the same kind of interrupt. Therefore, in order to continue to catch the same kind of interrupt, the code at the interrupt-handling address should reinvoke "cpint" before returning control to the point at which the interrupt occurred.

When the interrupt-handling address is 0 or any odd address, the operating system does not actually pass control to that address (see Arguments) and does not reset itself to take the default action the next time the same kind of interrupt occurs.

The file "/lib/sysints" defines the interrupts whose names are shown in the table accompanying this document.

If not caught or ignored, the default behavior of each program interrupt (except SIGDEAD and SIGDUMP) is to terminate the task to which it is sent. As shown in the table, some also produce a "core dump". A core dump is a file called "core" in the working directory which contains the task's image of the contents of memory. Each byte in the program and stack space is written to the core file immediately after receipt of the interrupt. The user can examine this file to determine the state of memory at the time the interrupt was received. A core file is often useful for diagnostic purposes. The operating system will not create a core file if the working directory contains a file named "core" which denies write permission to the current effective user or if the working directory denies write permission to the current effective user.

The default action for the SIGDUMP interrupt is to create a core dump and return control to the task. The task is not terminated.

A vendor may use a TRAP instruction with a number greater than 6. In such a case the user should not issue the instruction.

User-defined interrupts are available to the end user.

For further information on program interrupts see Section 6.4 of the 68xxx UniFLEX Programmer's Guide.

(continued)

Table 1. Table of Interrupts

Name	Number	Description	A	C	D	I	R
SIGHUP	1	Hangup	+	+	-	+	+
SIGINT	2	Keyboard	+	+	-	+	+
SIGQUIT	3	Quit	+	+	+	+	+
SIGEMT	4	A-line (Axxx) emulation trap	+	+	+	+	+
SIGKILL	5	Task kill	+	-	-	-	+
SIGPIPE	6	Broken pipe	+	+	-	+	+
SIGSWAP	7	Swap error	+	+	-	-	+
SIGTRACE	8	Trace	+	+	-	+	-
SIGTIME	9	Time limit	+	+	+	-	+
SIGALRM	10	Alarm	+	+	-	+	+
SIGTERM	11	Task terminate	+	+	-	+	+
SIGTRAPV	12	TRAPV instruction	+	+	+	+	+
SIGCHK	13	CHK instruction	+	+	+	+	+
SIGEMT2	14	F-line (Fxxx) emulation trap	+	+	+	+	+
SIGTRAP1	15	TRAP #1 instruction	+	+	+	+	+
SIGTRAP2	16	TRAP #2 instruction	+	+	+	+	+
SIGTRAP3	17	TRAP #3 instruction	+	+	+	+	+
SIGTRAP4	18	TRAP #4 instruction	+	+	+	+	+
SIGTRAP5	19	TRAP #5 instruction	+	+	+	+	+
SIGTRAP6	20	TRAP #6-14 instruction	+	+	+	+	+
SIGPAR	21	Parity error	+	+	+	-	+
SIGILL	22	Illegal instruction	+	+	+	-	+
SIGDIV	23	Division by 0	+	+	+	+	+
SIGPRIV	24	Privileged instruction	+	+	+	-	+
SIGADDR	25	Address error	+	+	+	-	+
SIGDEAD	26	A child task terminated	-	+	-	+	+
SIGWRIT	27	Write to read-only memory	+	+	+	-	+
SIGEXEC	28	Data or stack space violation	+	+	+	-	+
SIGBND	29	Segmentation violation	+	+	+	-	+
SIGUSR1	30	User-defined interrupt #1	+	+	-	+	+
SIGUSR2	31	User-defined interrupt #2	+	+	-	+	+
SIGUSR3	32	User-defined interrupt #3	+	+	-	+	+
SIGABORT	33	Program abort	+	-	-	-	+
SIGSPLR	34	Spooler signal	+	+	-	+	+
SIGINPUT	35	Input is ready	+	+	-	+	+
SIGDUMP	36	Take memory dump	0	+	+	+	+
	37-41	System-defined interrupts					
SIGUNORDERED	42	MC68881 branch or set on unordered operand	+	+	-	+	+
SIGINEXACT	43	MC68881 inexact result	+	+	-	+	+
SIGFPDIVIDE	44	MC68881 division by 0	+	+	-	+	+
SIGUNDERFLOW	45	MC68881 underflow	+	+	-	+	+
SIGOPERAND	46	MC68881 invalid operand	+	+	-	+	+
SIGOVERFLOW	47	MC68881 overflow	+	+	-	+	+
SIGSNAN	48	MC68881 signaling not-a-number	+	+	-	+	+
	49-63	Vendor-defined interrupts					

Notes: A = Default state is "abort" (otherwise, "ignore")
C = Interrupt can be caught
D = Produces a core dump
I = Interrupt can be ignored
R = Resets to default state when triggered
0 = See text

(continued)

NOTES

. The SIGTIME interrupt is not currently implemented.

ERRORS REPORTED

EBARG
The value of <interrupt> is not a valid interrupt number.

SEE ALSO

spint
Commands: qdb

create

Create a new file or truncate an existing file to a length of 0.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys create,<file_name>,<perm_mask>
```

Returns

<file_des> in D0

Arguments

<file_name> The address of the null-terminated name to assign to the file.

<perm_mask> An 8-bit mask specifying the permissions to assign to the file. The following table shows the permission that is assigned to the file as each bit is set. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

<perm_mask>	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users. When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

(continued)

DESCRIPTION

The "create" system call creates a new file or truncates an existing file to a length of 0. If the file already exists, the system ignores <perm_mask> and, instead, leaves the original permissions intact. In such a case the owner of the file also remains the same. The effective user must have write permission in the specified existing file in order for the system call to succeed.

If the file does not exist, the operating system creates a new file with the permissions specified by "anding" <perm_mask> with the 1's complement of the mask defining the default permissions for creating a file with the current task. The owner of the file is the effective user when the task invokes the system call. The effective user must have write permission in the parent directory of the new file in order for the system call to succeed.

In either case the "create" system call opens the specified file for writing whether or not the permissions grant such access to the effective user, and sets the current file position to the beginning of the file.

NOTES

- If the current task has the maximum permissible number of files open and the specified file does not exist, the "create" system call creates the file but does not open it.

ERRORS REPORTED

EDFUL
The device which was to contain the file has no file descriptor nodes (fdns) available.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

EPRM
The existing file or the parent of the new file does not grant the user write permission.

ETMFL
The current task already has open as many files as the operating system will allow. If the specified file does not already exist, the "create" system call creates it but cannot open it.

SEE ALSO

chacc
chprm
create_contiguous
defacc
open

create_contiguous

Create a new contiguous file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys create_contiguous,<file_name>,<perm_mask>,<file_size>,<zero_flag>
```

Returns

<file_des> in D0

Arguments

<file_name> The address of the null-terminated name to assign to the file.

<perm_mask> An 8-bit mask specifying the permissions to assign to the file. The following table shows the permission that is assigned to the file as each bit is set. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

<perm_mask>	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

<file_size> The number of bytes to reserve for the file. This number must be a multiple of 512.

<zero_flag> A flag which tells the operating system whether or not to fill the space allocated for the file with null bytes. Unless the specified value is 0, the operating system does so.

DESCRIPTION

The "create_contiguous" system call creates a new contiguous file. If the file already exists, the system ignores <perm_mask> and, instead, leaves the original permissions intact. In such a case the owner of the file also remains the same, and all information in the file at the time the system call is invoked is lost. The effective user must have write permission in the specified existing file in order for the system call to succeed.

If the file does not exist and the disk has enough unused contiguous-file space, the operating system creates a new file with the permissions specified by "anding" <perm_mask> with the 1's complement of the mask defining the default permissions for creating a file with the current task. The owner of the file is the effective user when the task invokes the system call. The effective user must have write permission in the parent directory of the new file in order for the system call to succeed.

In either case the "create_contiguous" system call opens the specified file for writing whether or not the permissions grant such access to the effective user, and sets the current file position to the beginning of the file.

NOTES

- If the current task has the maximum permissible number of files open and the specified file does not exist, the "create_contiguous" system call creates the file but does not open it.
- Contiguous files are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBARG
The value of <file_size> must be a multiple of 512.

EBDCL
The operating system does not support contiguous files.

EDFUL
The device which was to contain the file does not have enough contiguous-file space.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

EPRM
The existing file or the parent directory of the new file does not grant the user write permission.

ETMFL
The current task already has open as many files as the operating system will allow. If the specified file does not already exist, the "create_contiguous" system call creates it but cannot open it.

SEE ALSO

create
defacc

create_pty

Open an unused pseudoterminal channel.

ASSEMBLY LANGUAGE SYNTAX

Expected

<slave_file_des> in D0
<master_file_des> in A0

Syntax Statement

```
sys create_pty
```

Arguments

<slave_file_des>	The file descriptor for access to the pseudoterminal by a slave task.
<master_file_des>	The file descriptor for access to the pseudoterminal by a master task.

DESCRIPTION

The "create_pty" system call opens an unused pseudoterminal channel. A pseudoterminal is a mechanism which allows one program to communicate with another task as if it were communicating with a terminal. The operating system treats a pseudoterminal like a device. The task which creates the pseudoterminal is the "master task"; the task or tasks with which the master task communicates are the "slave tasks".

Before any task can open a channel to a pseudoterminal, the directory "/dev" must contain at least one pseudoterminal device. The system manager can create such a device with the "makdev" command as follows:

```
makdev /dev/pty<num> p 1 <num>
```

where <num> is a two-digit number between 00 and 99 inclusive. (The maximum number of pseudoterminals any system can support is 100; however, this maximum is system-dependent and may be less than 100 for any given system.) The numbers used to create all pseudoterminals on a system must be continuous and must start with 00. The "create_pty" system call returns access to the unused pseudoterminal with the lowest number.

The operating system opens a pseudoterminal with the same configuration as a terminal. That is, the pseudoterminal is not in raw mode; it does not map upper- to lowercase; it is not in single character input mode.

It does echo input, echo the backspace character, expand tabs on output, output a line-feed character (hexadecimal 0A) after each carriage return, and ignore control characters except for the carriage return, the horizontal tab character, control-C, control-D, control-\, the backspace character, and the line-cancel character. By default the backspace character is control-H; the line-cancel character, control-X. The master task or any slave task can alter these parameters with the "ttyset" system call. The configuration of the pseudoterminal reflects the most recent invocation of "ttyset". (For further details on the configuration of a terminal see the 68xxx UniFLEX Programmer's Guide).

The operating system creates a pseudoterminal with particular modes of operation in effect. This paragraph describes the default modes of operation. The operating system performs a "read" system call from the master task whether or not data are available. The "read" call from the master task returns only the data written by the slave task. Characters written from the master task to the pseudoterminal are processed normally, not as if the channel to the master task were in raw mode. The operating system completes a "write" system call from the master task before a slave task has consumed the data. Slave tasks may send data to the pseudoterminal. The state of all these modes can be altered with the "control_pty" system call.

Once a channel to a pseudoterminal is open, additional slave tasks may access that pseudoterminal by invoking the "open" system call for the appropriate device. All tasks using a pseudoterminal can treat it exactly as they would treat a terminal.

The "ofstat" system call may be used with a pseudoterminal. The call returns a status of FPMPTY for a master task and FPSPTY for a slave task.

NOTES

- Pseudoterminals are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBADF
The device specified is not a pseudoterminal.

EDFUL
All available pseudoterminal channels are already open. If the number of pseudoterminals in the device directory is less than the maximum the system can support, the system manager can create more of them.

SEE ALSO

control_pty
ofstat
ttyget
68xxx UniFLEX Programmer's Guide

crpipe

Create a pipe.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys crpipe
```

Returns

<read_file_des> in D0
<write_file_des> in A0

Arguments

<read_file_des>	The file descriptor for reading from the pipe.
<write_file_des>	The file descriptor for writing to the pipe.

DESCRIPTION

A UniFLEX pipe is a special kind of file which passes the output from one task to another task as input. The "crpipe" system call creates a pipe. Typically, a task invokes this system call before a "fork" system call so that the child task and its parent task can communicate. The "fork" system call duplicates both of the pipe's file descriptors for the child task. After the fork is complete, the task that is going to write to the pipe should close <read_file_des>, and the task that is going to read from it should close <write_file_des>.

The operating system sets the end-of-file condition for a pipe when the pipe is empty and no file descriptors for writing are open. It returns a "broken pipe" condition if the writing task tries to write to the pipe when no file descriptors for reading are open.

The operating system allows the task that is writing to the pipe to write up to 4,096 bytes before suspending that task. After the task that is reading from the pipe has read all of the data, the operating system wakes the first task and allows it to continue writing to the pipe. The operating system allows the task that is reading from the pipe to read whenever any data are in the pipe. If the task tries to read when the pipe is empty and a file descriptor for writing is open, the operating system puts the reading task to sleep until either the writing task sends data to the pipe or the end-of-file condition is set. If the writing task sends data to the pipe, the operating system wakes the reading task and allows it to read the data. If end-of-file is set,

the operating system wakes the task, and the "read" system call returns having read 0 bytes.

NOTES

- Because of the way in which the operating system determines the end-of-file and broken-pipe conditions, it is crucial for each task using the pipe to close the file descriptor it is not using. Failure to close the unused file descriptor for writing prevents the operating system from ever setting the end-of-file condition; failure to close the unused file descriptor for reading prevents the operating system from recognizing a broken pipe.

- Either task associated with the pipe may use the "ofstat" system call to determine whether or not the pipe contains any data. If the file size of the pipe is 0, no data are in the pipe. Otherwise, the pipe contains data.

- Either task associated with the pipe may use the "ofstat" system call to determine whether or not the other end of the pipe is closed. The value of the variable "st_cnt" returned by "ofstat" is the number of file descriptors open for the pipe. If this number is less than 2, one end of the pipe is closed.

ERRORS REPORTED

ETMFL
The current task has open more than two less than the maximum number of open files the operating system permits a single task. Opening the two file descriptors required for a pipe would exceed this maximum.

SEE ALSO

close
fork
ofstat
open
read
write

crtsd

Create a special file (a device) or a new directory.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys crtsd,<file_name>,<descriptor>,<dev_num>
```

Arguments

<file_name> The address of the null-terminated name of the new device or directory.

<descriptor> A 16-bit flag specifying the type of the new file and the access permissions to set for it. The low-order byte specifies the permissions. The following table shows the permission that is assigned to the file as each bit is set. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

<perm_mask>	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users. When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

The high-order byte specifies the type of file to create. The following table shows the valid bit patterns and the type of file each one creates:

Bit Pattern	Type of File
00000010	Block device
00000100	Character device
00000110	Pseudoterminal
00001000	Directory

<dev_num> A 16-bit flag containing the major and minor device number for a block device, a character device, or a pseudoterminal. The high-order byte contains the major device number; the low-order byte, the minor device number. The major device number tells the operating system which set of device drivers to use for the device; the minor device number indicates which particular physical device to associate with the specified file. If the device being created is a directory, the "crtsd" call ignores this argument.

DESCRIPTION

The "crtsd" creates a block device, a character device, a pseudoterminal device, or a new directory. A block device transfers and receives data one block (512 K) at a time; a character device, one character at a time. A pseudoterminal is a mechanism which allows one program to communicate with another task as if it were communicating with a terminal. A directory is a file that contains a series of entries, each one consisting of the name of a file and a pointer to the file descriptor node (fdn) for that file.

Only the system manager may invoke this system call.

ERRORS REPORTED

EBARG
Either <descriptor> or <dev_num> is invalid.

EDFUL
The device on which the user tried to create the special file is full.

EFLX
A file by the specified name already exists.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

EPRM
The current effective user is not the system manager.

SEE ALSO

create
create_pty

defacc

Set the default access permissions for all files created by this task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys defacc,<perm_mask>
```

Returns

<previous_mask> in D0

Arguments

<perm_mask> An 8-bit mask specifying the default permissions for all files created by the task invoking the "defacc" system call. The following table shows the permission that is assigned to the file as each bit is set. Only the low-order 6 bits are set by "defacc". The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

<perm_mask>	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users.

<previous_mask> The mask for default permissions that was in use prior to this invocation of the "defacc" system call.

(continued)

DESCRIPTION

The "defacc" system call sets the default access permissions for all files created by the task invoking the call. When the operating system executes the "create" system call, it "ands" the value of the 1's complement of the default permissions mask with the permissions mask specified by the user with the "create" system call. Thus, the operating system always disables permissions disabled by the default permissions mask regardless of the permission mask specified with "create".

A "fork" or "exec" system call passes its <perm_mask> to every task it spawns.

SEE ALSO

create
exec
fork
ofstat
stat
vfork

dup

Duplicate the specified file descriptor.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

```
sys dup
```

Returns

<new_file_des> in D0

Arguments

<file_des>	The file descriptor to duplicate.
<new_file_des>	The new file descriptor.

DESCRIPTION

The "dup" system call duplicates the specified file descriptor. To the user it appears that the operating system has again opened the file referenced by <file_des> in the same mode and with the same current position in the file.

ERRORS REPORTED

EBADF
The file descriptor <file_des> does not reference an open file.

EBARG
An argument to the system call is invalid.

ETMFL
The operating system already has open as many files as it can. The system manager can alter the number of open files allowed with the "tune" command up to the system-dependent maximum.

SEE ALSO

dups
open
Commands: tune

dups

Duplicate a file descriptor onto the specified file descriptor.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0
<requested_file_des> in A0

Syntax Statement

sys dups

Returns

<requested_file_des> in D0

Arguments

<file_des>	The file descriptor to duplicate.
<requested_file_des>	The file descriptor to use as the duplicate.

DESCRIPTION

The "dups" system call duplicates the specified file descriptor onto a specific file descriptor requested by the user. If the requested file descriptor references an open file, "dups" closes that file before proceeding. To the user it appears that the operating system has again opened the file referenced by <file_des> in the same mode and with the same current position in the file.

ERRORS REPORTED

EBADF
The file descriptor <file_des> does not reference an open file.

EBARG
One or both of the file descriptors specified are out of range.

SEE ALSO

dup

exec

Execute a program.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys exec,<file_name>,<arg_list>
```

Arguments

<file_name>	The address of the null-terminated name of the file containing the program to execute.
<arg_list>	The address of the list of addresses of the arguments to pass to the new program. Each argument must be a null-terminated character string. The list of arguments must be terminated by a null long-word.

DESCRIPTION

The "exec" system call tells the operating system to replace the program that is currently executing with the program found in the specified file. The operating system passes to the new program the arguments referenced by <arg_list> and begins executing the new program at its transfer address.

The operating system sets up the new program's stack so that it contains the number of arguments to the new program, the address of each argument, and the arguments themselves. Following is a picture of the stack as execution begins:

(continued)

```
... highest address in task space ...

        00
         .
         .
         .
argn: <argn>          (The last null-terminated argument string)
         .
         .
         .
arg0: <arg0>          (The first null-terminated argument string)
      00000000
      00000000
      argn            (A pointer to the last null-terminated
       .               argument string)
       .
       .
      arg0            (A pointer to the first argument string)
sp -> <argcnt>        (The number of arguments passed to the new
                       task--a 4-byte quantity)

... low memory ...
```

At least two null bytes are left at the top of the task's address space.

When the new program begins, it inherits the following attributes and resources from the calling program:

The task's priority
The task ID
The parent task's ID
The controlling terminal number
The mask specifying the default permissions for creating a file
The time remaining on an armed alarm clock
The working directory
All open files
System and user time information

If the user-ID bit is set on the file containing the calling program, the new program has as its effective user ID the ID of the owner of the file containing the new program. Otherwise, it inherits the user ID of the calling program.

The new task ignores any interrupts that the calling program ignored. All other interrupts sent to the new task result in their default action, which is usually termination of the task. The operating system disables profiling in the new program.

The "exec" system call returns to the calling program only if it fails.

ERRORS REPORTED

EARGC
The user specified too many arguments. The system allots approximately 4K for passing arguments.

EBBIG
The executable file is too large. The limit on the size of a file is machine-dependent.

EISDR
The specified file is a directory.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

ENOTB
The file containing the new program is not an executable file or cannot be executed on the hardware. For example, a program which contains 68020-specific instructions cannot be executed by a 68010-based machine.

EPRM
The permissions set on the specified file do not grant the requested type of access.

SEE ALSO

cpint
exece
fork
profile
vfork

exece

Execute a program.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys exece,<file_name>,<arg_list>,<env_list>
```

Arguments

<file_name>	The address of the null-terminated name of the file containing the program to execute.
<arg_list>	The address of the list of addresses of the arguments to pass to the new program. Each argument must be a null-terminated character string. The list of arguments must be terminated by a null long-word.
<env_list>	The address of the list of addresses of the environment parameters to pass to the new program. Each environment parameter must be a null-terminated character string. The list of environment parameters must be terminated by a null long-word.

DESCRIPTION

The "exece" system call tells the operating system to replace the program that is currently executing with the program found in the specified file. The operating system passes to the new program the arguments referenced by <arg_list> and the environment parameters referenced by <env_list>. It then begins executing the new program at its transfer address.

The operating system sets up the new program's stack so that it contains the number of arguments to the new program, the address of each argument, the arguments themselves, the address of each environment parameter, and the parameters themselves. Following is a picture of the stack as execution begins:

```
... highest address in task space ...

        00
         .
         .
         .
envn: <envn>          (The last null-terminated environment string)
         .
         .
         .
env0: <env>           (The first null-terminated environment string)
      00000000
argn: <argn>          (The last null-terminated argument string)
         .
         .
         .
arg0: <arg0>          (The first null-terminated argument string)
      00000000
      envn            (A pointer to the last null-terminated
       .               environment string)
       .
       .
      env0            (A pointer to the first null-terminated
                       environment string)
      00000000
      argn            (A pointer to the last null-terminated
       .               argument string)
       .
       .
      arg0            (A pointer to the first null-terminated
                       argument string)
sp -> <argcnt>        (The number of arguments passed to the new
                       task--a 4-byte quantity)

... low memory ...
```

At least two null bytes are left at the top of the task's address space.

When the new program begins, it inherits the following attributes and resources from the calling program:

The task's priority
The task ID
The parent task's ID
The controlling terminal number
The mask specifying the default permissions for creating a file
The time remaining on an armed alarm clock
The working directory
All open files
System and user time information

If the user-ID bit is set on the file containing the calling program, the new program has as its effective user ID the ID of the owner of the file containing the new program. Otherwise, it inherits the user ID of the calling program.

The new task ignores any interrupts that the calling program ignored. All other interrupts sent to the new task result in their default action, which is usually termination of the task. The operating system disables profiling in the new program.

The "exece" system call returns to the calling program only if it fails.

ERRORS REPORTED

EARGC
The user specified too many arguments. The system allots approximately 4K for passing arguments.

EBBIG
The executable file is too large. The limit on the size of a file is machine-dependent.

EISDR
The specified file is a directory.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

ENOTB
The file containing the new program is not an executable file or cannot be executed on the hardware. For example, a program which contains 68020-specific instructions cannot be executed by a 68010-based machine.

EPRM
The permissions set on the specified file do not grant the requested type of access.

SEE ALSO

cpint
exec
fork
profile
vfork

fcntl

Change or query the behavior of a file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des_1> in D0

Syntax Statement

sys fcntl,<function_code>

Returns

<access_flag> in D0
or
<file_des_2> in D0

Arguments

<file_des_1>	The file descriptor of the file whose behavior to change or query.
<function_code>	The subfunction to perform.
<access_flag>	A one-byte flag describing the mode of access to the file.
<file_des_2>	The file descriptor of the last file which sent the signal INPUT READY to the current task.

DESCRIPTION

The "fcntl" system call queries or changes the way in which a task may access the specified file. The method of access is defined in <access_flag>, which may be returned to D0 by "fcntl". When the operating system creates a file, none of these bits is set. Currently, the system uses only one bit of this flag, bit 0. By default, when a "read" system call tries to access a file, the operating system puts the task that issued the "read" call to sleep until data are available. When this bit is set, however, the operating system completes the system call whether or not data are available.

The "fcntl" system call supports four subfunctions, which are defined in the file "/lib/sysfcntl". The following table shows the function code associated with each of these subfunctions:

(continued)

Code	Subfunction
0	FCNTL_GET_PARAMS
1	not used
2	FCNTL_INPUT_FD
3	FCNTL_NOBLOCK
4	FCNTL_BLOCK

A description of each of these subfunctions follows:

FCNTL_GET_PARAMS	Return <access_flag> without altering any parameters.
FCNTL_INPUT_FD	Return the file descriptor of the last file which sent the signal INPUT READY. When the user specifies this subfunction, the "fcntl" system call ignores <file_des_1>.
FCNTL_NO_BLOCK	Complete any "read" system call that attempts to access the file whether or not data are available. Return the error ENOINPUT if no data are available, and send the signal INPUT READY to the task when data become available.
FCNTL_BLOCK	If no data are available, put to sleep until data become available any task issuing a "read" system call that attempts to access the specified file.

NOTES

. The INPUT READY signal is not sent to a task until a request to read from a file in NOBLOCK mode has been unsuccessful due to insufficient data.

ERRORS REPORTED

EBADF
The value of <file_des_1> does not reference an open file.

EBARG
The subfunction code is invalid.

filtim

Change the time of last modification of the specified file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<time> in D0

Syntax Statement

```
sys filtim,<file_name>
```

Arguments

<time>	The number of seconds that elapsed between midnight (00:00), January 1, 1980, and the desired last time of modification.
<file_time>	The address of the null-terminated name of the file to alter.

DESCRIPTION

The "filtim" system call changes the time of last modification of the specified file. It does not compare the new time of modification to either the time the file was created or the current time, so it is possible to set the file's modification time to a time before its creation or to a time in the future.

Only the system manager may invoke this system call.

ERRORS REPORTED

EBARG
An argument to the system call is invalid.

EBSY
The specified file is currently open.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
Either the file is on a device that is mounted for reading only or the current effective user is not the system manager.

SEE ALSO

ofstat
stat

fork

Create a new task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys fork
```

Returns

```
To parent task: <child_task's_ID> in D0
To child task:  0 in D0
```

Arguments

<child_task's_ID> The task ID assigned to the child task.

DESCRIPTION

The "fork" system call creates a new task (the child task) that is a copy of the current task (the parent task). The child task has the same priority, user ID, effective user-ID, controlling terminal information, default permissions-mask, working directory, signal handling set-up, and profiling information as the parent task. However, it differs from the parent task in the following ways: its task ID is different; its parent task-ID is the task ID of the parent task; its data are the same, but they are located in a different place in memory; its file descriptors are the same, but they are located in a different place in memory; its system and user CPU times are set to 0; its alarm clock is turned off.

After a "fork" system call the child task resumes executions at the instruction immediately following the "fork" call. The parent task, on the other hand, resumes execution 2 bytes after the "fork" call. Obviously, then, the first instruction in the new task must be a short branch (requiring only 2 bytes). Each task determines where to resume by looking at the contents of the D0 register immediately after execution of the "fork" call.

ERRORS REPORTED

ETMTS
: Either the maximum number of tasks allowed to a user or the maximum number of tasks allowed to the operating system has been reached. The system manager can alter either or both of these limits with the "tune" command up to the system-dependent maxima.

(continued)

EVFORK
The current task shares its memory with its parent and may not invoke this system call.

SEE ALSO

vfork

FPU_exception

Return or update information about an exception generated by the MC68881 floating-point coprocessor.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys FPU_exception,<function_code>,<buf_add>
```

Arguments

<function_code>	The subfunction to perform. A value of 0 tells the operating system to copy the exception information from its location in memory into the specified buffer; a value of 1, to copy the exception information in the buffer back to its location in memory.
<buf_add>	The address of the buffer for the exception information.

DESCRIPTION

In order for the "FPU_exception" system call to function, the user must have enabled exception processing by setting the appropriate bits in the MC68881 control register. A usable subset of exception processing can be enabled by setting a bit in the header of the binary file after assembly with the "headset" command. This subset is system-dependent, but in general it includes processing for exceptions generated by division by 0, overflow, and operand errors. A user who wishes to define exception processing directly from the assembly language program may do so with the "fmove" or "fmovem" instruction.

If exception processing is enabled, the "FPU_exception" system call either copies the exception information from its location in memory into the specified buffer or copies the exception information in the buffer back to its location in memory.

The file "/lib/sys68881" defines the structure of the buffer as follows:

(continued)

```
     * Buffer structure

      struct 0

     FPUstate  ds.l  46  State frame
     FPCR      ds.l   1  Control register
     FPSR      ds.l   1  Status register
     FPIAR     ds.l   1  Interrupt address register
     FPregs    ds.x   8  Data registers

     CPU_registers    ds.l  0
     CPU_D_registers  ds.l  8  Data registers
     CPU_A_registers  ds.l  8  Address registers

     CPU_stack_frame  ds.w  0
     CPU_SR           ds.w  1  Status register
     CPU_PC           ds.l  1  Program counter
     CPU_SFT          ds.w  1  Stack frame type
     CPU_PCX          ds.l     Program counter
     CPU_IR           ds.w  1  Internal register
     CPU_OP           ds.w  1  Operation word
     CPU_EA           ds.l     Effective address

     FPU_INTERRUPT_DATA_SIZE ds.w 0

     * Definitions for control registers

     * Symbols ending in "_b" are bit numbers.
     * Symbols ending in "_bf" are bit-field specifications.

     FPCR_BSUN_b     equ    15  Branch or set on unordered operand
     FPCR_SNAN_b     equ    14  Signaling Not-a-Number
     FPCR_OPERR_b    equ    13  Invalid operand
     FPCR_OVFL_b     equ    12  Overflow
     FPCR_UNFL_b     equ    11  Underflow
     FPCR_DZ_b       equ    10  Division by 0
     FPCR_INEX2_b    equ     9  Inexact result generated by an operation
     FPCR_INEX1_b    equ     8  Inexact result generated by decimal input
     FPCR_PREC_bf  bfequ  24:2  Rounding precision (see below)
     FPCR_RND_bf   bfequ  26:2  Rounding mode (see below)

     * Rounding precisions (FPCR_PREC_bf)

     PREC_EXTENDED  equ  0  Extended precision
     PREC_SINGLE    equ  1  Single precision
     PREC_DOUBLE    equ  2  Double precision
```

```
* Rounding modes (FPCR_RND_bf)

RND_TO_NEAREST  equ  0  Round toward nearest
RND_TO_ZERO     equ  1  Round toward zero
RND_TO_MINUS    equ  2  Round toward minus infinity
RND_TO_PLUS     equ  3  Round toward plus infinity


* Definitions for status registers

* Symbols ending in "_b" are bit numbers.
* Symbols ending in "_bf" are bit-field specifications.

FPSR_N_b            equ   27  Negative
FPSR_Z_b            equ   26  Zero
FPSR_I_b            equ   25  Infinity
FPSR_NAN_b          equ   24  Not-a-Number or unordered
FPSR_S_b            equ   23  Sign of quotient
FPSR_QUOTIENT_bf  bfequ  9:7  Seven least-significant bits of quotient
FPSR_BSUN_b         equ   15  Branch or set on unordered operand
FPSR_SNAN_b         equ   14  Signaling Not-a-Number
FPSR_OPERR_b        equ   13  Invalid operand
FPSR_OVFL_b         equ   12  Overflow
FPSR_UNFL_b         equ   11  Underflow
FPSR_DZ_b           equ   10  Division by 0
FPSR_INEX2_b        equ    9  Inexact result generated by an operation
FPSR_INEX1_b        equ    8  Inexact result generated by decimal input
FPSR_IOP_b          equ    7  Invalid operation
FPSR_AOVFL_b        equ    6  Accrued overflow
FPSR_AUNFL_b        equ    5  Accrued underflow
FPSR_ADZ_b          equ    4  Accrued division by 0
FPSR_INEX_b         equ    3  Accrued inexact
```

NOTES

- The user should use the "FPU_resume" system call to resume execution of the interrupted MC68881 instruction and to exit from the exception-handling routine. Using the "rtr" instruction may lead to unpredictable results because the program counter stored on the stack may be incorrect.

ERRORS REPORTED

ENOFPUDATA
No exception information is available.

(continued)

SEE ALSO

FPU_resume
Commands: headset
68xxx UniFLEX Relocating Assembler and Linking-Loader
Motorola, 1985. MC68881 Floating-Point Coprocessor User's Manual.
Austin: Motorola.

FPU_resume

Resume execution of the MC68881 instruction that generated an exception.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys FPU_resume
```

DESCRIPTION

The "FPU_resume" system call resumes execution of an MC68881 instruction that generated an exception. It should be used at the end of an exception-handling routine instead of the "rtr" instruction. Use of the "rtr" instruction to exit from an exception-handling routine may lead to unpredictable results because the program counter stored on the stack may be incorrect.

ERRORS REPORTED

ENOFPUDATA
No exception information is available.

SEE ALSO

FPU_exception

gpid

Get the task ID of the parent of the current task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys gpid
```

Returns

<parent_task_ID> in D0

Arguments

<parent_task_ID>	The task ID of the parent of the current task.

DESCRIPTION

The "gpid" system call gets the task ID of the parent of the current task and returns it to the D0 register.

SEE ALSO

gtid

gtid

Get the task ID of the current task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys gtid
```

Returns

<task_ID> in D0

Arguments

<task_ID> The task ID of the current task.

DESCRIPTION

The "gtid" system call gets the task ID of the current task and returns it to the D0 register.

SEE ALSO

exec
fork
gpid
vfork

guid

Return the user ID and the effective user ID of the person executing the current task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys guid
```

Returns

```
<user_ID> in D0
<effective_user_ID> in A0
```

Arguments

<user_ID>	The user ID of the person who is logged in on the terminal from which the current task is being run.
<effective_user_ID>	The user ID that defines the access permissions of the current task. If the task's user-ID bit is set, the effective user ID and the user ID may not be the same.

DESCRIPTION

The "guid" system call returns the user ID and the effective user ID of the person executing the current task.

SEE ALSO

suid

ind

Execute the system call located at the specified address.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys ind,<call_address>
```

Arguments

<call_address>	The address at which the system call to execute is located.

DESCRIPTION

The "ind" system call passes control to the specified address, which contains the system call to execute. This call is useful when the values of the arguments to the desired system call are not known prior to execution of the program. The user allocates space for the arguments in the assembly language program, and during execution the program moves the appropriate values to the proper location.

The code located at <call_address> may invoke neither an "ind" nor an "indx" system call.

ERRORS REPORTED

EBDCL
: Either the code located at <call_address> does not invoke a valid UniFLEX system call, or it invokes an "ind" or "indx" system call.

SEE ALSO

indx

indx

Execute the system call located at the specified address.

ASSEMBLY LANGUAGE SYNTAX

Expected

<call_address> in A0

Syntax Statement

sys indx

Arguments

<call_address>	The address at which the system call to execute is located.

DESCRIPTION

The "indx" system call passes control to the address in the A0 register, which contains the system call to execute. It is similar to the "ind" system call, but it allows the system call and its arguments to be located anywhere in memory, including the stack. This call is useful when the values of the arguments to the desired system call are not known prior to execution of the program. During execution the program moves the appropriate values to the proper location.

The code located at <call_address> may invoke neither an "indx" nor an "ind" system call.

ERRORS REPORTED

EBDCL
Either the code located at <call_address> does not invoke a valid UniFLEX system call, or it invokes an "ind" or "indx" system call.

SEE ALSO

ind

link

Create a link to a file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys link,<file_name_1>,<file_name_2>
```

Arguments

<file_name_1>	The address of the null-terminated name of an existing file. The file may not be a directory unless the current effective user is the system manager.
<file_name_2>	The address of the null-terminated name of the file to link to <file_name_1>. This file must be nonexistent at the time the user invokes the system call.

DESCRIPTION

The "link" system call establishes a link between the specified existing file, <file_name_1>, and <file_name_2>. After the link is created, any reference to <file_name_2> references <file_name_1>. Creation of the link does not alter the original file in any way except that the time of last modification is updated to the time the link was created.

The operating system cannot link a file on one device to a file on another device.

The directory containing the new link must grant write permission to the current effective user. The user must also have execute permission in all but the last component of both specified file names.

NOTES

- The maximum link count is 127. More than 127 links may exist, but the count itself neither increases nor decreases once it reaches 127. Therefore, once a link count reaches 127, neither the "kill" command nor the "unlink" system call can remove the file from the system because the link count cannot reach 0. The removal of the last link will, therefore, result in an unreferenced file, which can be deleted by the "diskrepair" command.

(continued)

ERRORS REPORTED

EFLX
The file <file_name_2> already exists.

EISDR
The file <file_name_1> is a directory, but the current effective user is not the system manager.

EMSDR
The path to either <file_name_1> or <file_name_2> or both cannot be followed.

ENDR
A part of the path to one or both of the specified files is not a directory.

ENOFL
No file on the system corresponds to <file_name_1>.

EPRM
Either the directory to contain the new link, <file_name_2>, does not grant write permission to the effective user, or the user does not have execute permission in all but the last component of both specified file names.

EXDEV
The specified files are not on the same device; therefore, the system cannot link them.

SEE ALSO

unlink
Commands: diskrepair, kill

lock

Lock a task in memory or unlock a locked task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
lock,<function_code>
```

Arguments

<function_code>	A number indicating the subfunction to perform. A nonzero value tells the operating system to lock the task in memory; a value of 0, to unlock it.

DESCRIPTION

The "lock" system call locks a task in memory or unlocks a locked task. The operating system cannot take memory from a locked task to use for another task.

Only the system manager may invoke this system call.

NOTES

. Unlocking a task that is not locked is not an error.

ERRORS REPORTED

EPRM
The current effective user is not the system manager.

SEE ALSO

memman

lrec

Add an entry to the operating system's lock table.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys lrec,<count>

Arguments

<file_des>	The file descriptor for the file containing the record to lock.
<count>	The number of bytes to lock from the current position in the file.

DESCRIPTION

The "lrec" system call adds to the operating system's lock table an entry for the open file referenced by <file_des>. If the current task already has an entry for that file descriptor in the system's lock table, "lrec" removes that entry. If another task has an entry for the file descriptor in the system's lock table, and that entry contains all or part of the record the current task is trying to lock, the system call fails.

Locking a record only prevents other users from locking any of the data in that record. It does not prevent anyone from reading or modifying the record or any other part of the file.

The operating system removes all entries a task makes in the lock table when the task terminates.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file, or it references a pipe, a character device, a block device, or a pseudoterminal.

EBARG
An argument to the system call is invalid.

ELOCK
The specified record cannot be locked either because all or part of the record is already locked by another task or because the system's lock table is full. The system manager may use the "tune" command to alter the number of entries the lock table can contain.

SEE ALSO

urec

make_realtime

Make a non-real-time task a real-time task and set its relative priority, or make a real-time task a non-real-time task.

ASSEMBLY LANGUAGE SYNTAX

Expected

<relative_priority> in D0

Syntax Statement

```
sys make_realtime
```

Arguments

<relative_priority>	A value used by the system scheduler to set the relative priority of a real-time task in case it must schedule the CPU among several real-time tasks. The value must be in the range of -25 to 25. The operating system ignores a value which is outside this range. A value of 0 tells the system scheduler that the task is no longer a real-time task.

DESCRIPTION

The "make_realtime" system call either makes a non-real-time task a real-time task or makes a real-time task a non-real-time task. It also sets the priority of a real-time task in case the system scheduler must share the CPU among more than one real-time task. The priority of a real-time task is fixed by the "make_realtime" or the "setpr" system call. Any real-time task has a higher priority than any non-real-time task. Only a real-time task of higher priority can usurp the CPU from a real-time task.

If the system call is used to make a real-time task a non-real-time task (the user specifies a relative priority of 0), the operating system sets the priority of that task as it would normally.

Only the system manager may invoke this system call.

NOTES

. Real-time tasks are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBDCL
The system does not support real-time tasks.

EPRM
The current effective user is not the system manager.

SEE ALSO

setpr
yield_CPU

memman

Perform a memory-management operation.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys memman,<function_code>,<start_address>,<end_address>
```

Arguments

<function_code>	The subfunction to perform.
<start_address>	The first address in the region of memory to control (but see DESCRIPTION). The value of <start_address> must be less than the value of <end_address).
<end_address>	The last address in the region of memory to control (but see DESCRIPTION). The value of <end_address> must be greater than the value of <start_address).

DESCRIPTION

The "memman" system call performs a memory-management operation on the specified region of memory. It supports five subfunctions. The following table shows the function code associated with each of these functions. Some subfunctions, however, may not be available on a particular implementation.

Code	Subfunction
0	Clear the region's "dirty bit". When set, the dirty bit tells the operating system that the copy of the page in the paging space is out of date.
1	Lock the region in memory.
2	Unlock the region from memory.
3	Write protect the region.
4	Remove write protection from the region.
5	Release the memory allocated to the region.

Regardless of the addresses specified, the region that the "memman" system call acts on is a set of pages. Each page contains 4K of memory. By default, the region ranges from the beginning of the page containing <start_address> to the end of the page containing <end_address>. For example, if the user specifies $1020 as <start_address> and $10A0 as <end_address>, "memman" acts on the region from $1000 through $1FFF inclusive. However, if the user adds 32 to the function code, the

region that "memman" acts on includes only those complete pages within the specified range. The addresses specified in the previous example do not encompass a complete page. Therefore, if the function code is in the range of 32 through 36 inclusive, they specify a null range. A <start_address> of $0040 and an <end_address> of $2FFF with a function code between 32 and 36 specify the region from $1000 through $2FFF inclusive.

ERRORS REPORTED

EBARG
The function code is invalid; <start_address> is greater than <end_address>; or the address range specified by <start_address> and <end_address> is outside the task's address space.

EVFORK
The current task shares its memory with its parent and may not invoke this system call.

SEE ALSO

lrec

mount

Insert the medium in the specified block device at the node of the directory tree specified by <dir_name>.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys mount,<dev_name>,<dir_name>,<mode>
```

Arguments

<dev_name>	The address of the null-terminated name of the device containing the medium to mount. The specified device must be a block device.
<dir_name>	The address of the null-terminated name of the directory on which to mount the specified device.
<mode>	A value indicating whether or not to write-protect the mounted medium. A value of 0 specifies to mount the medium for reading and writing; a nonzero value, for reading only.

DESCRIPTION

The "mount" system call temporarily inserts the medium in the specified block device at the node of the directory tree specified by <dir_name>. As long as the medium is mounted, any references to <dir_name> actually access the root directory of the mounted medium. If the medium is mounted for reading and writing, the "mount" function sets an indicator on the medium indicating that it is currently mounted. The "unmnt" system call clears this indicator.

Any files in the directory on which the device is mounted are inaccessible for the duration of the mount.

Only the system manager may invoke this system call.

NOTES

- Disks written by the "backup" command cannot be mounted.

ERRORS REPORTED

EBSY
: Either the operating system's mount table is full or something is already mounted on <dir_name>.

EDIRTY
The last time the medium in the specified device was mounted, it was not properly unmounted. It may, therefore, be corrupt. The user should try to salvage the data by executing the "diskrepair" command.

EFLX
The medium in the specified device is already mounted.

EIO
The operating system cannot read the data on the medium in the specified device; no medium is in the specified device; or the medium is not correctly formatted. If "mount" returns this error when a properly formatted medium is in the device, the user should try to salvage the data on the medium by executing the "diskrepair" command.

EMSDR
The path to either <dev_name> or <dir_name> or both cannot be followed.

ENBLK
The specified device is not a block device.

ENDR
Either <dir_name> is not a directory or a part of the path to <dev_name> or <dir_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The current effective user is not the system manager.

EWRTPROT
The specified device is write-protected.

SEE ALSO

crtsd
unmnt
Commands: backup, diskrepair

msg_attach

Attach a task to a message exchange.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys msg_attach,<exchange_name>,<mode>
```

Returns

<exchange_ID> in D0

Arguments

<exchange_name> The 4-byte name of the message exchange to attach to the task. The name may contain any unique value, for example four ASCII characters.

<mode> The access mode to use. A value of 0 indicates that the task is to send messages to the exchange; a value of 1, that it is to receive messages from the exchange. The file "/lib/sysmessages" defines the following constants:

Value	Constant
0	MSG_send_mode
1	MSG_receive_mode

<exchange_ID> An identification number to associate with this particular exchange. All other system calls referencing the exchange use this ID rather than the name of the exchange.

DESCRIPTION

A message exchange is a section of memory set aside as a repository for messages. Such exchanges allow intertask communication. An individual task can attach to no more than 32 message exchanges at the same time. The number of exchanges the system can support is system-dependent. The system manager can adjust this number within the permissible limits with the "tune" command.

The "msg_attach" system call attaches a message exchange to the current task. The first attempt to attach to a nonexistent exchange in receive mode creates the exchange. The operating system does not allow a task to attach to a nonexistent exchange in send mode.

NOTES

- Message exchanges are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EDFUL
: The operating system is already using the maximum permissible number of message exchanges. The system manager can alter this limit with the "tune" command up to the system-dependent maximum.

ENOFL
: No message exchange on the system corresponds to the specified name, and the mode is MSG_Send_mode.

ETMFL
: The task attempted to attach to more than 32 message exchanges at the same time.

SEE ALSO

msg_detach
msg_send
msg_receive
msg_status
Commands: tune

msg_detach

Detach a task from a message exchange.

ASSEMBLY LANGUAGE SYNTAX

Expected

<exchange_ID> in D0

Syntax Statement

```
sys msg_detach
```

Arguments

<exchange_ID>	The identification number of the exchange to detach from. This ID must be one returned from an "msg_attach" system call.

DESCRIPTION

The "msg_detach" system call detaches the task from a message exchange. After the last task attached to a message exchange has been detached from it, the exchange is released to the system. It may then be reallocated with the "msg_attach" system call.

NOTES

- Message exchanges are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBADF
The value of <exchange_ID> does not correspond to any message exchange which is attached to the task.

SEE ALSO

msg_attach
msg_send
msg_receive
msg_status

msg_receive

Receive a message from another task via a message exchange.

ASSEMBLY LANGUAGE SYNTAX

Expected

<exchange_ID> in D0

Syntax Statement

sys msg_receive,<buf_add>,<mode>

Arguments

<exchange_ID>	The identification number of the exchange from which to fetch a message. This ID must be one returned from an "msg_attach" system call.
<buf_add>	The address of the buffer to receive the text of the message. This buffer should contain at least as many characters as are used for messages in the system. By default, the size of messages on any system is 64 bytes. The system manager can change this value with the "tune" command.
<mode>	The mode to use while fetching the message. A value of 0 tells the operating system to return an error if no messages are at the specified message exchange. A value of 1 tells the operating system to suspend the receiving task until a message has been queued at the exchange. The file "/lib/sysmessages" defines the following constants:

Value	Constant
0	MSG_no_wait_for_messages
1	MSG_wait_for_messages

DESCRIPTION

The "msg_receive" system call receives a message from another task via a message exchange. The operating system writes the first message in the queue associated with the specified exchange to the buffer at <buf_add>.

NOTES

- Message exchanges are a vendor-dependent option and may not be supported by all systems.

- Because the size of messages on any system is fixed, the operating system always moves the same number of bytes into the message buffer. If the buffer is not large enough to hold the message, the operating system simply writes past the end of the buffer, destroying any information located there. The destruction of such information could result in an address error, which would cause the program to abort.

ERRORS REPORTED

EBADF
The value of <exchange_ID> does not correspond to any message exchange which is attached to the task.

EINTR
The task caught an interrupt, which caused the system call to end abnormally.

ENOINPUT
No messages are available at the exchange, and the task did not elect to wait for one.

SEE ALSO

msg_attach
msg_detach
msg_send
msg_status
Commands: tune

msg_send

Send a message to another task via a message exchange.

ASSEMBLY LANGUAGE SYNTAX

Expected

<exchange_ID> in D0

Syntax Statement

sys msg_send,<buf_add>,<mode>

Arguments

<exchange_ID> The identification number of the exchange to which to send a message. This ID must be one returned from an "msg_attach" system call.

<buf_add> The address of the buffer containing the text of the message to send. This buffer should contain at least as many characters as are used for messages in the system. By default, the size of messages on any system is 64 bytes. The system manager can change this value with the "tune" command.

<mode> A 2-bit mask specifying the mode to use while sending the message. The following table shows the correspondence between the values of <mode> and the mode in which to operate. Any combination of bits is valid.

Value	Constant
00	MSG_no_wait
01	MSG_wait_for_consumption
10	MSG_wait_for_space

When MSG_no_wait mode is in effect, the operating system does not wait for a task to consume the message. Nor does it wait for space to send a message if the system is saturated with messages (rather, it returns an error). If MSG_wait_for_consumption is in effect, the operating system suspends the sending task until some other task consumes this particular message. If MSG_wait_for_space is in effect, the operating system suspends the sending task until the

system is not saturated with messages.

DESCRIPTION

The "msg_send" system call sends a message to another task via a message exchange. The operating system writes the text in the buffer at <buf_add> to the queue for the specified message exchange.

The operating system limits the total number of messages that can be queued at all message exchanges. The system manager can adjust this limit with the "tune" command. A user can avoid this limit by invoking the command in MSG_wait_for_space mode.

NOTES

- Message exchanges are a vendor-dependent option and may not be supported by all systems.

- Because the size of messages on any system is fixed, the operating system always moves the same number of bytes from the beginning of the message buffer to the appropriate message exchange. If the buffer is not as large as this fixed size, the operating system simply uses information located past the end of the buffer. Obviously if such information is used, the latter part of the message will probably not make sense.

ERRORS REPORTED

EBADF
The value of <exchange_ID> does not correspond to any message exchange which is attached to the task.

EINTR
The task caught an interrupt, which caused the system call to end abnormally. The operating system returns this error if a task invokes the system call when MSG_wait_for_consumption or MSG_wait_for_space mode is in effect and receives an interrupt while it is waiting. In such a case the system call aborts with no ill effects. Thus, by using the waiting modes in conjunction with an alarm interrupt, the task may specify the maximum amount of time to wait.

ENOINPUT
The maximum number of messages allowed in all queues has been reached. The system manager can alter this limit with the "tune" command up to the system-dependent maximum.

SEE ALSO

msg_attach
msg_detach
msg_receive
msg_status
Commands: tune

msg_status

Obtain information about the status of a message exchange.

ASSEMBLY LANGUAGE SYNTAX

Expected

<exchange_ID> in D0

Syntax Statement

```
sys msg_status,<buf_add>
```

Arguments

<exchange_ID>	The identification number of the exchange to report on. This ID must be one returned from an "msg_attach" system call.
<buf_add>	The address of the buffer to contain the information about the status of the message exchange. The buffer should contain at least MBX_STAT_SIZE characters, where MBX_STAT_SIZE is defined in the file "/lib/sysmessages".

DESCRIPTION

The "msg_status" system call obtains information about the status of a message exchange and writes that information to the buffer specified by <buf_add>. The information includes the name of the message exchange, the number of tasks currently attached to the exchange as senders, the number of tasks currently attached to the exchange as receivers, the number of messages in the queue, and the size of each message.

The structure of this buffer is defined in the file "/lib/sysmessages" as follows:

```
* Structure of buffer for status of exchange

 struct 0

msg_name      ds.l 1  Name of exchange
msg_senders   ds.w 1  Number of tasks attached as senders
msg_receivers ds.w 1  Number of tasks attached as receivers
msg_q_count   ds.w 1  Number of messages in the queue
msg_size      ds.w 1  Size of each message on the system

MBX_STAT_SIZE ds.w 0
```


(continued)

The size of all messages on a system is fixed, but varies from system to system because the system manager can adjust the size with the "tune" command. The "msg_status" system call returns the value of the size so that a task may adjust its buffers to conform to the value on the system in use.

NOTES

- Message exchanges are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBADF
The value of <exchange_ID> does not correspond to any message exchange which is attached to the task.

SEE ALSO

msg_attach
msg_detach
msg_receive
msg_send
Commands: tune

ofstat

Get the status of an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys ofstat,<buf_add>

Arguments

<buf_add> The address of the buffer to contain the information on the status of the file.

DESCRIPTION

The "ofstat" system call writes to <buf_add> the information describing the status of the file referenced by <file_des>. The file "/lib/sysstat" defines the structure of this buffer as follows:

```
* Definition of buffer for "status" and "ofstat"

 base 0 Set initial values

st_dev  ds.w  1  Device number
st_fdn  ds.w  1  Fdn number
        ds.b  1  Filler
st_mod  ds.b  1  File mode
st_prm  ds.b  1  Permission bits
st_cnt  ds.b  1  File link count
st_own  ds.w  1  File owner's user ID
st_siz  ds.l  1  File size in bytes
st_mtm  ds.l  1  Time of file's last modification
st_spr  ds.b  4  Spare--for future use only

ST_SIZ  ds.w  0  Size of status buffer
```

The value "st_dev" is the device number of the device containing the file referenced by <file_des>; "st_fdn" is the file descriptor number (fdn) of the specified file. The variable "st_mod" is an 8-bit mask describing the type of the file; the low-order bit is ignored. The following table shows the valid values for this mask and the type of file associated with each value. The file "/lib/sysstat" defines the

constants whose names are shown in parentheses.

st_mod	Type of File
0000000x	Regular file (FSREG)
0000001x	Block device (FSBLK)
0000010x	Character device (FSCHR)
0000100x	Directory (FSDIR)
1000011x	Master pseudoterminal (FSMPTY)
0000011x	Slave pseudoterminal (FSSPTY)
0100000x	Pipe (FSPIPE)

The variable "st_prm" is an 8-bit mask describing the access permissions for the file. The following table shows the type of permission that is associated with each bit in the mask. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

st_prm	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users. When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

The value of "st_cnt" is the number of links to the specified file, or in the case of a pipe, the number of file descriptors open for the pipe; "st_own" is the user ID of the owner of the file; "st_siz" is the number of bytes in the file, or in the case a of a pipe, a flag indicating whether or not the pipe contains any data (if the file size of the pipe is 0, no data are in the pipe); "st_mtm" is the time (expressed as the number of seconds that had elapsed since midnight (00:00), January 1, 1980) the file was last modified; "st_spr" is currently unused.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file.

EBARG
An argument to the system call is invalid.

SEE ALSO

status

open

Open an existing file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys open,<file_name>,<mode>
```

Returns

<file_des> in D0

Arguments

<file_name>	The address of the null-terminated name to assign to the open file.
<mode>	A value which tells the operating system what sort of access to allow to the file. If <mode> is 0, the operating system opens the file for reading only; if 1, for writing only; if 2, for both reading and writing. If <file_name> references a character device, the user may add hexadecimal 8000 to this value to specify exclusive access. Exclusive access, which is only supported by some character devices, is useful, for example, in the case of a port that is used both as a terminal and for communications.
<file_des>	A number by which all other system calls must reference the open file.

DESCRIPTION

The "open" system call opens the specified file in the mode described by <mode>, sets the current file position to the beginning of the file, and assigns a file descriptor to the file.

ERRORS REPORTED

EBARG
The value for <mode> is invalid.

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The file's access permissions do not grant the user the type of access requested by <mode>.

ETMFL
The current task already has open as many files as the operating system will allow. The system manager can alter this limit with the "tune" command up to the system-dependent maximum.

SEE ALSO

close
create
Commands: tune

phys

Obtain or release access to a section of system memory.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys phys,[-]<code>
```

Returns

<log_base_add> in D0 (only when obtaining access to memory)

Arguments

-	Release access to the specified section of memory. In the absence of a minus sign the system obtains access to the specified section of memory.
<code>	A number specifying the section of system memory to obtain or release access to. The correspondence between <code> and a particular system resource is part of the configuration of the system. A value of 0 releases all sections of memory allocated by the current task through the "phys" system call.
<log_base_add>	The logical address of the base of a resource that is mapped into the task's address space.

DESCRIPTION

The "phys" system call accesses or releases the resource specified by <code>. If the specified resource has already been allocated by the task, "phys" ignores a request to allocate it. The system call also ignores any request to release a resource that has not been allocated to the task.

ERRORS REPORTED

EBARG
The value of <code> is invalid.

SEE ALSO

Commands: tune

profile

Start or stop monitoring the current task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys profile,<start_add>,<buf_add>,<size>,<scale>
```

Arguments

<start_add>	The address at which to begin monitoring the task.
<buf_add>	The address of the buffer to contain the results of the monitoring.
<size>	The number of bytes in the buffer at <buf_add>.
<scale>	A number determining the granularity of the monitoring procedure. A value of 0 or 1 tells the operating system to stop monitoring the task. Other valid values are 2, 4, 8, 16, 32, 64, and 128. If the user specifies an invalid value, "profile" uses the largest valid value that does not exceed the specified value.

DESCRIPTION

The "profile" system call starts or stops monitoring the current task. If the value of <scale> is 0 or 1, monitoring stops; otherwise, it begins.

While monitoring a task the operating system examines the task at each tick of the system clock, which occurs every tenth of a second. If the program counter is at an address less than <start_add>, the operating system does nothing; otherwise, it subtracts the value of <start_add> from the value of the current program counter, divides the result by <scale>, and multiplies the quotient by 2. If this product is less than <size>, the operating system adds its value to <buf_add> and increments the word at the resulting address by 1. If the product is greater than <size>, no change is made in the buffer at <buf_add>.

NOTES

- The operating system automatically stops monitoring a task when that task invokes any "exec" system call, but not when it invokes the "fork" or "vfork" system call.

SEE ALSO

exec
exece
fork

read

Read data from an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys read,<buf_add>,<count>

Returns

<bytes_read>

Arguments

<file_des>	The file descriptor of the open file to read.
<buf_add>	The address of the buffer to contain the data that are read.
<count>	The number of bytes to read. The system call performs more efficiently if this number is greater than 512 and less than 4,096.

DESCRIPTION

The "read" system call reads data from the open file referenced by <file_des>. It begins reading at the current file position and continues until it has read <count> bytes, has reached the end of the file, or, if the specified file is a terminal, has read an end-of-line character. It writes the data it reads into the buffer located at <buf_add>. If the specified file is one that the user can randomly access, "read" sets the current file position to the byte immediately following the last byte read.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file, or the file is not open for reading.

EBARG
The value for either <count> or <file_des> is invalid.

EINTR
The task received and caught an interrupt while the function was reading from a slow device.

EIO
The operating system returned an I/O error. In such a case the data in the buffer may not be the same as the data in the file.

SEE ALSO

create
crpipe
dup
dups
open
write

rump

Create, destroy, access, or relinquish access to a named resource.

ASSEMBLY LANGUAGE SYNTAX

Expected

<function_code> in D0
<resource_name> in A0

Syntax Statement

```
sys rump
```

Arguments

<function_code>	The subfunction to perform.
<resource_name>	The name of the resource to act on. It must be a null-terminated character string containing between 2 and 16 characters (including the null character).

DESCRIPTION

The "rump" (resource utilization management protocol) system call provides a means of supplying exclusive access to physical resources such as I/O devices and special shared memory. It does so by allowing the user to create a named resource associated with a particular physical resource. The association is purely in the mind of the user. Because the operating system does not recognize the association between the physical resource and the named resource, any user who does not know of the existence of the named resource or who chooses to ignore its existence can access the device directly. All users should, therefore, be advised of the existence of any named resources so that they can honor the protocol of asking for access to those resources before using them.

The number of named resources that a system can support is system-dependent.

The "rump" system call supports four subfunctions. The following table shows the function code associated with each of these subfunctions:

Code	Subfunction
1	RUMP_ENQUEUE
2	RUMP_DEQUEUE
3	RUMP_CREATE
4	RUMP_DESTROY

A description of each of these subfunctions follows:

RUMP_ENQUEUE	Obtain exclusive access to the specified named resource. If another task currently has access to the resource, the operating system puts the calling task to sleep until the resource becomes available. If more than one task is waiting for access to a resource, the operating system provides access in the order in which the requests were made.
RUMP_DEQUEUE	Relinquish access to the specified named resource.
RUMP_CREATE	Create a named resource. This subfunction does not provide access to the resource.
RUMP_DESTROY	Destroy the specified named resource. Any user, not only the user who created it, may destroy a resource. A resource cannot, however, be destroyed while any user has access to it.

NOTES

- Named resources are a vendor-dependent option and may not be supported by all systems.

ERRORS REPORTED

EBADF
The user tried to relinquish access to a named resource but did not have access to the specified resource.

EBSY
The user tried to destroy a named resource, but the specified resource was in use.

EDFUL
The user tried to create a named resource, but the system is already supporting as many resources as it can.

EFLX
The specified named resource already exists.

EINTR
The user tried to obtain access to a named resource, but the "rump" system call terminated abnormally. The operating system returns this error if a task invokes the RUMP_ENQUEUE subfunction on a busy resource and receives an interrupt while it is waiting for the resource. In such a case the subfunction aborts with no ill effects. Thus, by using the RUMP_ENQUEUE subfunction in conjunction with an alarm interrupt, the task may specify the maximum amount of time to wait for a busy resource.

ENOFL
The specified named resource does not exist.

sacct

Enable or disable system accounting procedures.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys sacct,<file_name>
```

Arguments

<file_name>	The address of the null-terminated name of the file to contain the data collected by the accounting process. The specified file must already exist. A null address ($0000) tells the operating system to disable the accounting procedures.

DESCRIPTION

The "sacct" system call enables or disables the system's accounting procedures. When the accounting procedures are enabled, the operating system appends a record to the specified file each time a task terminates. Each record is of the following structure, which is defined in the file "/lib/sysacct". A tick is one one-hundredth of a second.

```
 *  Structure for accounting record

  struct 0

 acuid    ds.b  2  User ID
 acstrt   ds.b  4  Starting time of task
 acend    ds.b  4  Ending time of task
 acsyst   ds.b  3  System time used by task (in ticks)
 acusrt   ds.b  3  User time used by task (in ticks)
 acstat   ds.b  2  Termination status of task
 actty    ds.b  1  Number of terminal where task originated
 acmem    ds.b  1  Maximum memory used by task (4K blocks)
 acblks   ds.b  2  I/O units used (measure of blocks read or written)
 acspar   ds.b  2  Spare
 acname   ds.b  8  Name of command executed by task

 AC_SIZ   ds.b  0  Size of accounting record
```

Only the system manager may invoke this system call.

ERRORS REPORTED

EBADF
The specified file is not a regular file.

EFLX
The system accounting procedures are already enabled.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The current effective user is not the system manager.

seek

Change the current file position of an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

```
sys seek,<count>,<pt_of_origin>
```

Returns

<new_position>

Arguments

<file_des>	The file descriptor of the file to reposition.
<count>	A four-byte, signed number specifying the number of bytes to shift the current file position from the point of origin (see following argument).
<pt_of_origin>	A value indicating where in the file to begin the specified shift. A value of 0 specifies the beginning of the file; of 1, the current position in the file; of 2, the end of the file.
<new_position>	The new current file position, which is expressed as the number of bytes beyond the beginning of the file. The first byte in the file is byte 0.

DESCRIPTION

The "seek" system call changes the current file position in the file referenced by <file_des>. If the specified position is beyond the current end of the file, the system sets the current file position accordingly, but does not actually allocate any new blocks if the file resides on a block device. A "read" system call returns null bytes for the data in the resulting gap in the file.

(continued)

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file or the file is not open in the proper mode.

EBARG
An argument to the system call is invalid.

ESEEK
Either the requested file position is before the beginning of the file or the file descriptor references a file which cannot be randomly accessed, such as a pipe.

SEE ALSO

create
dup
dups
open

setpr

Change the priority bias of the current task.

ASSEMBLY LANGUAGE SYNTAX

Expected

<priority_bias> in D0

Syntax Statement

```
sys setpr
```

Arguments

<priority_bias>	If the current task is a non-real-time task, <priority_bias> is a bias used by the system scheduler for scheduling the sharing of the CPU among several non-real-time tasks. If the task is a real-time task, <priority_bias> is a value used by the scheduler to set the relative priority of that task in case it must schedule the sharing of the CPU among several real-time tasks. In all cases the value of <priority_bias> must be in the range of -25 to 25 inclusive. The operating system ignores a value which is outside this range. Only the system manager may use a negative number.

DESCRIPTION

The "setpr" system call changes the priority bias of the current non-real-time task or sets the priority of a real-time task.

The priority of a non-real-time task may range from -128 to 127, with a task with a priority of -128 having the highest priority on the system; one with 127, the lowest. The system determines the priority by summing two values: a dynamic value, which it calculates based on what the task is currently doing, and a static value, the priority bias, which is initially 0. If the sum of these two components is less than -128, the operating system sets the priority to -128; if greater than 127, to 127. Two non-real-time tasks with the same priority alternate their use of the CPU.

The priority of a real-time task does not include a dynamic component; rather, it is fixed by the "setpr" or the "make_realtime" system call. Any real-time task has a higher priority than any non-real-time task. Only a real-time task of higher priority can usurp the CPU from a real-time task.

ERRORS REPORTED

EPRM
The value specified for <priority_bias> is negative, but the current effective user is not the system manager.

SEE ALSO

make_realtime
yield_CPU

set_high_address_mask

Load the specified value into the register for the hardware address mask.

ASSEMBLY LANGUAGE SYNTAX

Expected

<address_mask> in D0

Syntax Statement

```
sys set_high_address_mask
```

Arguments

<address_mask>	A 32-bit value used to mask the high-order byte of all addresses from the microprocessor. The default is 0xFFFFFFFF. The user must supply a 32-bit value, but only the high-order byte is significant. The operating system sets bits 0 through 23 inclusive.

DESCRIPTION

The "set_high_address_mask" system call loads the specified value into the register for the hardware address mask.

NOTES

- The operating system supports the "set_high_address_mask" system call only on the Tektronix 4406.

spint

Send an interrupt to a task.

ASSEMBLY LANGUAGE SYNTAX

Expected

<task_ID> in D0

Syntax Statement

```
sys spint,<interrupt>
```

Arguments

<task_ID>	The task ID of the task to interrupt. A value of 0 tells the operating system to send the interrupt to all tasks associated with the caller's controlling terminal. If <task_ID> is -1 and the current effective user is the system manager, the system sends the interrupt to all tasks on the system except 0 and 1, which are the scheduler and the initializer.
<interrupt>	The kind of interrupt to send (see table).

DESCRIPTION

The "spint" interrupt sends the specified interrupt to the task whose ID is <task_ID> if that task has the same effective user as the task sending the interrupt or if the effective user of the calling task is the system manager. The file "/lib/sysints" defines the interrupts whose names are shown in the table accompanying this document.

If not caught or ignored, the default behavior of each program interrupt (except SIGDEAD and SIGDUMP) is to terminate the task to which it is sent. As shown in the table, some also produce a "core dump". A core dump, which is a file in the working directory called "core", contains an image of the contents of the task's memory. Each byte in the program and stack space is written to a disk file immediately after receipt of the interrupt. The user can examine this file to determine the state of memory at the time the interrupt was received. A core dump is often useful for diagnostic purposes. The operating system will not create such a file if the working directory contains a file named "core" which denies write permission to the current effective user or if the working directory denies write permission to the current effective user.

Table 1. Table of Interrupts

Name	Number	Description	A	C	D	I	R
SIGHUP	1	Hangup	+	+	-	+	+
SIGINT	2	Keyboard	+	+	-	+	+
SIGQUIT	3	Quit	+	+	+	+	+
SIGEMT	4	A-line (Axxx) emulation trap	+	+	+	+	+
SIGKILL	5	Task kill	+	-	-	-	+
SIGPIPE	6	Broken pipe	+	+	-	+	+
SIGSWAP	7	Swap error	+	+	-	-	+
SIGTRACE	8	Trace	+	+	-	+	-
SIGTIME	9	Time limit	+	+	+	-	+
SIGALRM	10	Alarm	+	+	-	+	+
SIGTERM	11	Task terminate	+	+	-	+	+
SIGTRAPV	12	TRAPV instruction	+	+	+	+	+
SIGCHK	13	CHK instruction	+	+	+	+	+
SIGEMT2	14	F-line (Fxxx) emulation trap	+	+	+	+	+
SIGTRAP1	15	TRAP #1 instruction	+	+	+	+	+
SIGTRAP2	16	TRAP #2 instruction	+	+	+	+	+
SIGTRAP3	17	TRAP #3 instruction	+	+	+	+	+
SIGTRAP4	18	TRAP #4 instruction	+	+	+	+	+
SIGTRAP5	19	TRAP #5 instruction	+	+	+	+	+
SIGTRAP6	20	TRAP #6-14 instruction	+	+	+	+	+
SIGPAR	21	Parity error	+	+	+	-	+
SIGILL	22	Illegal instruction	+	+	+	-	+
SIGDIV	23	Division by 0	+	+	+	+	+
SIGPRIV	24	Privileged instruction	+	+	+	-	+
SIGADDR	25	Address error	+	+	+	-	+
SIGDEAD	26	A child task terminated	-	+	-	+	+
SIGWRIT	27	Write to read-only memory	+	+	+	-	+
SIGEXEC	28	Data or stack space violation	+	+	+	-	+
SIGBND	29	Segmentation violation	+	+	+	-	+
SIGUSR1	30	User-defined interrupt #1	+	+	-	+	+
SIGUSR2	31	User-defined interrupt #2	+	+	-	+	+
SIGUSR3	32	User-defined interrupt #3	+	+	-	+	+
SIGABORT	33	Program abort	+	-	-	-	+
SIGSPLR	34	Spooler signal	+	+	-	+	+
SIGINPUT	35	Input is ready	+	+	-	+	+
SIGDUMP	36	Take memory dump	0	+	+	+	+
	37-41	System-defined interrupts					
SIGUNORDERED	42	MC68881 branch or set on unordered operand	+	+	-	+	+
SIGINEXACT	43	MC68881 inexact result	+	+	-	+	+
SIGFPDIVIDE	44	MC68881 division by 0	+	+	-	+	+
SIGUNDERFLOW	45	MC68881 underflow	+	+	-	+	+
SIGOPERAND	46	MC68881 invalid operand	+	+	-	+	+
SIGOVERFLOW	47	MC68881 overflow	+	+	-	+	+
SIGSNAN	48	MC68881 signaling not-a-number	+	+	-	+	+
	49-63	Vendor-defined interrupts					

Notes: A = Default state is "abort" (otherwise, "ignore")
C = Interrupt can be caught
D = Produces a core dump
I = Interrupt can be ignored
R = Resets to default state when triggered
0 = See text

(continued)

The default action for the SIGDUMP interrupt is to create a core dump and return control to the task. The task is not terminated.

A vendor may use a TRAP instruction with a number greater than 6. In such a case the user should not issue the instruction.

User-defined interrupts are available to the end user.

NOTES

- A controlling terminal is normally the terminal that started the task or one of the task's ancestors. A task can change its controlling terminal by closing all files that refer to terminals and then opening a terminal device. This device becomes the controlling terminal.

- The SIGTIME interrupt is not currently implemented.

ERRORS REPORTED

EBARG
The value of <interrupt> is not a valid interrupt number.

ENTSK
The value of <task_ID> is not a valid task ID.

EPRM
The current effective user is neither the system manager nor the current effective user of the specified task.

SEE ALSO

cpint

stack

Extend the stack space of the current task.

ASSEMBLY LANGUAGE SYNTAX

Expected

<address> in A0

Syntax Statement

sys stack

Arguments

<address> The address to which to extend the stack. If <address> is greater than the address that is currently the end of the stack, the task relinquishes to the operating system all memory up to and including <address>.

DESCRIPTION

The "stack" system call extends the stack space of the current task. Initially the operating system assigns between 100 and 3,000 bytes to the stack, depending on the number of arguments passed with the "exec" system call.

ERRORS REPORTED

ESTOF
The stack space of the current task is as large as it can get. The user can change the maximum size of the program with the "headset" command.

EVFORK
The current task shares its memory with its parent and may not invoke this system call.

SEE ALSO

break
Commands: headset

stack_limit

Specify a limit to the task's stack segment.

ASSEMBLY LANGUAGE SYNTAX

Expected

<address> in A0

Syntax Statement

sys stack_limit

Returns

<previous_limit> in D0

Arguments

<address>	The desired lowest address for the stack. The operating system truncates the specified value to the address of the first byte in the page containing <address>. Specifying an address of 0 removes the limit on the size of the stack. If the specified address is in a page that is higher than the page currently referenced by the stack pointer, the operating system immediately sends the task a SIGEXEC interrupt.
<previous_limit>	The value of the previous stack limit. The system call returns 0 if no previous stack limit was in effect.

DESCRIPTION

The "stack_limit" system call sets a lower limit on the growth of the program's stack segment. The specified value is truncated to the lowest address in the page. If the operating system attempts to allocate a page of memory for the stack which is below the limit, the memory is not allocated and a SIGEXEC interrupt is sent to the task.

NOTES

- The operating system checks the stack limit only when processing the "stack" system call or expanding the task's logical address space as part of the automatic growth of the stack. If the memory page

immediately below the stack limit is already part of the task's logical address space the limit is checked only when processing the "stack" system call. If the user is raising the stack limit, it is good programming practice to issue a "stack" system call with the same argument as passed to the "stack_limit" system call in order to remove those stack pages that are below the stack limit from the task's logical address space.

ERRORS REPORTED

EBARG
The specified address is larger than the maximum valid value for the stack.

SEE ALSO

stack

status

Get the status of a file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys status,<file_name>,<buf_add>
```

Arguments

<file_name>	The address of the null-terminated name of the file to examine.
<buf_add>	The address of the buffer to contain the information on the status of the file.

DESCRIPTION

The "status" system call writes to <buf_add> the information describing the status of the file referenced by <file_name>. The file "/lib/sysstat" defines the structure of this buffer as follows:

```
* Definition of buffer for "status" and "ofstat"

 base 0 Set initial values

st_dev  ds.w  1  Device number
st_fdn  ds.w  1  Fdn number
        ds.b  1  Filler
st_mod  ds.b  1  File mode
st_prm  ds.b  1  Permission bits
st_cnt  ds.b  1  File link count
st_own  ds.w  1  File owner's user ID
st_siz  ds.l  1  File size in bytes
st_mtm  ds.l  1  Time of file's last modification
st_spr  ds.b  4  Spare--for future use only

ST_SIZ  ds.w  0  Size of status buffer
```

The value of "st_dev" is the device number of the device containing the file referenced by <file_des>; "st_fdn" is the file descriptor number (fdn) of the specified file. The variable "st_mod" is an 8-bit mask describing the type of the file; the low-order bit is ignored. The following table shows the valid values for this mask and the type of file associated with each value. The file "/lib/sysstat" defines the constants whose names are shown in parentheses.

st_mod	Type of File
0000000x	Regular file (FSREG)
0000001x	Block device (FSBLK)
0000010x	Character device (FSCHR)
0000100x	Directory (FSDIR)
1000011x	Master pseudoterminal (FSMPTY)
0000011x	Slave pseudoterminal (FSSPTY)
0100000x	Pipe (FSPIPE)

The variable "st_prm" is an 8-bit mask describing the access permissions for the file. The following table shows the type of permission that is associated with each bit in the mask. The file "/lib/sysstat" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

st_prm	Permission Assigned
00000001	Owner read permission (FACUR)
00000010	Owner write permission (FACUW)
00000100	Owner execute permission (FACUE)
00001000	Others read permission (FACOR)
00010000	Others write permission (FACOW)
00100000	Others execute permission (FACOE)
01000000	Set user-ID bit for execute (FXSET)

If the specified file is a directory, FACUE grants permissions to the user who owns the file to search the directory for the name of a file; FACOE grants the same permission to other users. When the user-ID bit is set, the operating system grants to any user who executes the file the same permissions as it grants to the owner of the file for the duration of the task.

The value of "st_cnt" is the number of links to the specified file; "st_own" is the user ID of the owner of the file; "st_siz" is the number of bytes in the file; "st_mtm" is the time (expressed as the number of seconds that had elapsed since midnight (00:00), January 1, 1980) the file was last modified; "st_spr" is currently unused.

ERRORS REPORTED

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

SEE ALSO

ofstat

stime

Set the system date and time.

ASSEMBLY LANGUAGE SYNTAX

Expected

<time> in D0

Syntax Statement

```
sys stime
```

Arguments

<time> The number of seconds that have elapsed since midnight (00:00), January 1, 1980, at the zeroth meridian (Greenwich, England).

DESCRIPTION

The "stime" system call sets the system time and date to the value specified by <time>.

Only the system manager may invoke this system call.

NOTES

- The operating system converts <time> to the system time based on its perception of the time zone, which by default is 300 minutes west of Greenwich, England. The system manager may use the "tune" command to alter the time zone.

ERRORS REPORTED

EPRM
The current effective user is not the system manager.

SEE ALSO

Commands: date, tune

stop

Suspend the current task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys stop
```

DESCRIPTION

The "stop" system call suspends the current task indefinitely. The system call returns only if the task receives an interrupt, catches it, and returns from the interrupt-handling routine. If "stop" does return, it always returns the error EINTR.

ERRORS REPORTED

EINTR
: If the "stop" system call returns, it returns this error.

SEE ALSO

alarm
cpint
spint

suid

Set both the user ID and the effective user ID.

ASSEMBLY LANGUAGE SYNTAX

Expected

<user_ID> in D0

Syntax Statement

```
sys suid
```

Arguments

<user_ID>	The user ID to assign to both the user and the effective user.

DESCRIPTION

The "suid" system call sets the user ID of both the user and the effective user. At the time the task invokes the system call, either the user or the effective user must be the system manager.

ERRORS REPORTED

EPRM
Neither the user nor the effective user is the system manager.

SEE ALSO

guid

term

Terminate the current task.

ASSEMBLY LANGUAGE SYNTAX

Expected

<term_status> in D0

Syntax Statement

sys term

Arguments

<term_status>	The termination status to assign to the task as it terminates. A nonzero status should indicate an error.

DESCRIPTION

The "term" system call terminates the current task. It never returns to the caller. The termination status is available to the parent task through the "wait" system call.

SEE ALSO

wait

time

Get the system time and other related parameters.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys time,<buf_add>
```

Arguments

<buf_add> The address of the buffer to contain the information obtained by the system call.

DESCRIPTION

The "time" system call returns the system time and several related parameters to the specified buffer. The file "/lib/systim" defines the structure of the buffer at <buf_add> as follows:

```
* Buffer for "time"

 base 0

tm_sec  ds.l  1  Time in seconds since 00:00, January 1, 1980
tm_tik  ds.b  1  Number of ticks into the current second
tm_dst  ds.b  1  Flag for Daylight Savings Time
tm_zon  ds.w  1  Time zone

TM_SIZ  ds.w  0  Size of buffer
```

The operating system stores the time as a 4-byte value, "tm_sec", which represents the number of seconds that have elapsed since midnight (00:00), January 1, 1980, at the zeroth meridian (Greenwich, England). The value in "tm_tik" represents the number of ticks that had passed in the current second when the system call was invoked. Each tick represents one one-hundredth of a second.

The Daylight Savings Time flag, "tm_dst" indicates whether or not Daylight Savings Time is observed locally. A value of 0 indicates that it is not; a value of 1, that it is. The default value is 0. The operating system assumes that Daylight Savings Time begins and ends according to the dates used in the United States--the last Sunday in April and the last Sunday in October. The system manager may alter the value of "tm_dst" with the "tune" command.

The value stored in "tm_zon" represents the time difference in minutes between local time and Universal Time. A positive value of "tm_zon" indicates the number of minutes west of Greenwich, England; a negative value, the number of minutes east. The default value is 300. The system manager may alter the value of "tm_zon" with the "tune" command.

SEE ALSO

Commands: tune

truncate

Set the size of an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

```
sys truncate
```

Arguments

<file_des>	The file descriptor of the file whose size to alter. It must reference a regular file.

DESCRIPTION

The "truncate" system call sets the size of the file referenced by <file_des> so that its end-of-file is the current file position. The file must be open for writing. If the current file position is before the existing end-of-file, the system call truncates the file. If the current file position is beyond the existing end-of-file, the system call extends the file. When "truncate" extends a file, it does not actually allocate any new blocks if the file resides on a block device. A "read" system call returns null bytes for the data in the resulting gap in the file.

NOTES

- If <file_des> references a device or a pipe, the "truncate" system call does nothing.

ERRORS REPORTED

EBADF
: The file descriptor does not reference an open file, or the file is not open for writing.

EBARG
: An argument to the system call is invalid.

EISDR
The file referenced by <file_des> is a directory.

SEE ALSO

seek

ttime

Get the information on the use of the CPU by the current task and its child tasks.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys ttime,<buf_add>
```

Arguments

<buf_add> The address of the buffer to contain the information obtained by the system call.

DESCRIPTION

The "ttime" system call obtains information about the use of the central processing unit (CPU) by the current task, by the operating system on behalf of the current task, by all child tasks, and by the operating system on behalf of all child tasks. The operating system continuously updates the information about usage of the CPU by the current task and by the operating system on behalf of the current task. It updates the information about usage of the CPU by child tasks and by the operating system on behalf of all child tasks whenever a child task terminates.

The following table shows the structure of the buffer at <buf_add> as defined in the file "/lib/systim". All numbers represent ticks. A tick is one one-hundredth of a second.

```
* Buffer for "ttime"

 base 0

ti_usr  ds.l  1  CPU use by current task
ti_sys  ds.l  1  CPU use by system on behalf of current task
ti_chu  ds.l  1  CPU use by all descendants of current task
ti_chs  ds.l  1  CPU use by system on behalf of all descendants

TM_SIZ  ds.w  0  Size of buffer for "ttime"
```

SEE ALSO

fork
vfork

ttyget

Get information on the configuration of a terminal.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys ttyget,<buf_add>

Arguments

<file_des>	The file descriptor of the file associated with the terminal to examine.
<buf_add>	The address of the buffer to contain the information about the configuration of the specified terminal.

DESCRIPTION

The "ttyget" system call obtains information about the configuration of the terminal referenced by <file_des> and writes that information into the buffer at <buf_add>. The file "/lib/systty" defines the structure of this buffer as follows:

```
* Definition of buffer for "ttyset" and "ttyget"

 base 0

tt_flg  ds.b  1  Flags
tt_dly  ds.b  1  Delays
tt_cnc  ds.b  1  Line-cancel character (default is control-X)
tt_bks  ds.b  1  Backspace character (default is control-H)
tt_spd  ds.b  1  Terminal speed
tt_spr  ds.b  1  Stop output byte

TT_SIZ  ds.w  0  Size of buffer
```

The information contained in the first byte of the buffer, "tt_flg", describes the way in which the terminal handles input and output. The following table shows the type of behavior governed by each bit. The file "/lib/systty" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

Bit Pattern	Behavior
00000001	Raw I/O mode (RAW)
00000010	Echo input (ECHO)
00000100	Expand tabs on output (XTABS)
00001000	Map upper- to lowercase (LCASE)
00010000	Automatic line-feed (CRMOD)
00100000	Echo backspace-echo character (BSECH)
01000000	Single-character input mode (SCHR)
10000000	Ignore control characters

These modes are described in detail in Section 6.2 of the 68xxx UniFLEX Programmer's Guide.

The information contained in the second byte of the buffer, "tt_dly", defines the length of the delay the system uses after outputting a new-line character, a carriage return, a tab character, a vertical tab character, or a form-feed character. A delay is useful in cases where a slow output device such as a teleprinter, which requires a delay for carriage returns, is attached to the system. The following table shows the kind of delay implemented by each bit. Any combination of bits is valid although the system ignores bits 6 and 7.

Bit Pattern	Kind of Delay Affected	Length of Delay (msec)
00000001	New-line	10
00000010	New-line	20
00000100	Carriage return	10
00001000	Carriage return	20
00010000	Tab	20
00100000	Vertical tab	240

The file "/lib/systty" defines certain combinations of bits in the delay byte as follows:

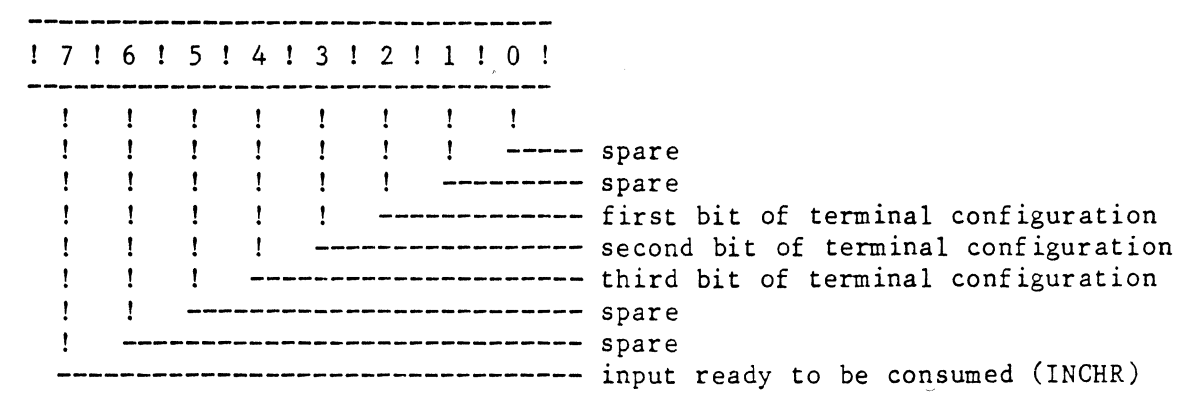
Bit Pattern	Constant	Kind of Delay	Length of Delay (msec)
00000011	DELNL	New-line	30
00001100	DELCR	Carriage return	30
00010000	DELTB	Tab	20
00100000	DELVT	Vertical tab	240
00100000	DELFF	Form-feed	240

The third and fourth bytes of the buffer define the line-cancel and backspace characters. The default line-cancel character is control-X; the default backspace, control-H.

(continued)

The fifth byte of the "ttyset" buffer is the terminal speed byte. This byte presently implements only four bits. Bits 2, 3, and 4 define the configuration of the terminal; bit 7 is a flag which, when set, indicates that the terminal has input characters waiting to be consumed by the program. This bit is only meaningful when read--that is, the input-ready condition should not be set with the "ttyset" system call. A picture of this byte follows. The file "/lib/systty" defines the constant INCHR.

Terminal speed byte (tt_spd):



Under normal input operations the "input ready to be consumed" bit does not come on until an entire line has been input and terminated by a carriage return. There are special input modes which can be established, however, where the "input ready to be consumed" bit will come on as soon as a single character is input. These modes, "raw I/O mode" and "single character input mode" are described in Section 6.2 of the 68xxx UniFLEX Programmer's Guide.

The following table shows the configuration of the terminal for all possible settings of the terminal configuration bits:

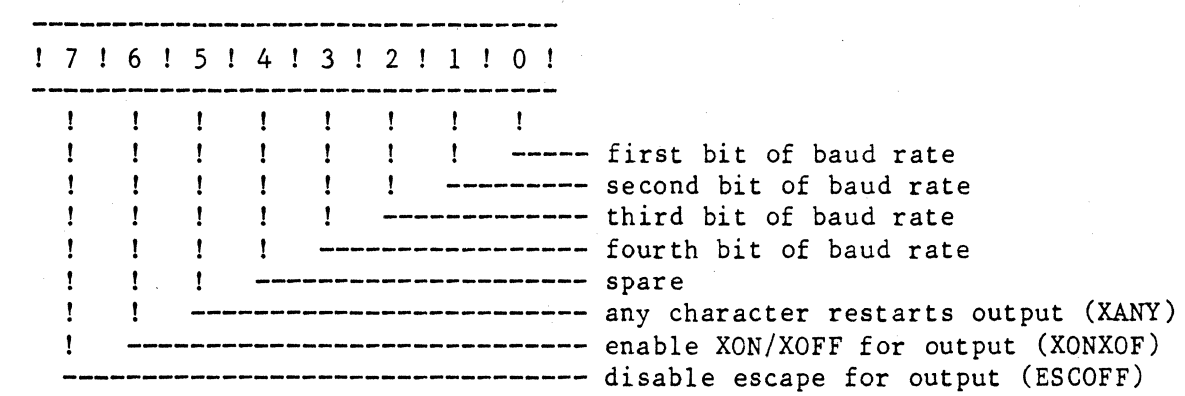
Terminal Configuration (Bit Pattern)	Data Bits	Stop Bits	Parity
0 0 0	7	2	Even
0 0 1	7	2	Odd
0 1 0	7	1	Even
0 1 1	7	1	Odd
1 0 0	8	2	None
1 0 1	8	1	None
1 1 0	8	1	Even
1 1 1	8	1	Odd

(continued)

The last byte in the "ttyset" buffer is the stop output byte, which defines the baud rate and which characters may be used to stop and start output. A user may stop and start output to a terminal by one of two methods: using the escape key or using XON/XOFF processing.

The escape key method permits a user to type an escape character (hexadecimal 1B) to stop output. A subsequent escape character restarts the output. The XON/XOFF method permits a user to type an XOFF character (hexadecimal 13) to stop output and a subsequent XON character (hexadecimal 11) to restart it. Many terminals produce XON and XOFF characters automatically to prevent the computer from sending too many characters to the terminal at once. The escape and XON/XOFF mechanisms can be independently enabled or disabled by setting or clearing the appropriate bits in the byte "tt_spr". A picture of the byte follows. The file "/lib/systty" defines the constants whose names are shown in parentheses.

Stop output byte (tt_spr):



If XANY is in effect, the terminal drivers restart output stopped by either an escape or an XOFF character when the user types any character.

The following table shows the baud rate defined by all possible settings of the first four bits of the stop output byte:

Bit Pattern	Baud Rate	Bit Pattern	Baud Rate
0 0 0 0	--	1 0 0 0	1200
0 0 0 1	75	1 0 0 1	1800
0 0 1 0	110	1 0 1 0	2400
0 0 1 1	134.5	1 0 1 1	3600
0 1 0 0	150	1 1 0 0	4800
0 1 0 1	200	1 1 0 1	7200
0 1 1 0	300	1 1 1 0	9600
0 1 1 1	600	1 1 1 1	19200

(continued)

Not all hardware supports all of these baud rates, and not all hardware allows the dynamic changing of baud rates.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file.

EBARG
An argument to the system call is invalid.

ENTTY
The file referenced by <file_des> is not a character device.

SEE ALSO

ttyset
68xxx UniFLEX Programmer's Guide

ttynum

Get the terminal number of the task's controlling terminal.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys ttynum
```

Returns

<tty_num> in D0

Arguments

<tty_num>	The number of the task's controlling terminal. The controlling terminal is normally the terminal that started the task or one of the task's ancestors. A task can change its controlling terminal by closing all files that refer to terminals and then opening a terminal device. This device becomes the controlling terminal.

DESCRIPTION

The "ttynum" system call returns the number of the task's controlling terminal. If the task has no controlling terminal (standard input, standard output, and standard error are all closed), the system call returns 0. Of course, "ttynum" also returns 0 if the controlling terminal is "/dev/tty00", the console. It is impossible to distinguish between these two cases.

ttyset

Set the configuration of a terminal.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys ttyset,<buf_add>

Arguments

<file_des>	The file descriptor of the file associated with the terminal to alter.
<buf_add>	The address of the buffer containing the information about how to set the configuration of the specified terminal.

DESCRIPTION

The "ttyset" system call reads the information in the buffer at <buf_add> and changes the configuration of the terminal referenced by <file_des> accordingly. Normally, the user should invoke the "ttyget" system call prior to the "ttyset" system call in order to obtain information about the current configuration of the terminal. The desired bits should be set or cleared using the logical operators "and" and "or". The file "/lib/systty" defines the structure of this buffer as follows:

```
* Definition of buffer for "ttyset" and "ttyget"

 base 0

tt_flg  ds.b  1  Flags
tt_dly  ds.b  1  Delays
tt_cnc  ds.b  1  Line-cancel character (default is control-X)
tt_bks  ds.b  1  Backspace character (default is control-H)
tt_spd  ds.b  1  Terminal speed
tt_spr  ds.b  1  Stop output byte

TT_SIZ  ds.w  0  Size of buffer
```

The information contained in the first byte of the buffer, "tt_flg", describes the way in which the terminal handles input and output. The

following table shows the type of behavior governed by each bit. The file "/lib/systty" defines the constants whose names are shown in parentheses. Any combination of bits is valid.

Bit Pattern	Behavior
00000001	Raw I/O mode (RAW)
00000010	Echo input (ECHO)
00000100	Expand tabs on output (XTABS)
00001000	Map upper- to lowercase (LCASE)
00010000	Automatic line-feed (CRMOD)
00100000	Echo backspace-echo character (BSECH)
01000000	Single-character input mode (SCHR)
10000000	Ignore control characters

These modes are described in detail in Section 6.2 of the 68xxx UniFLEX Programmer's Guide.

The information contained in the second byte of the buffer, "tt_dly", defines the length of the delay the system uses after outputting a new-line character, a carriage return, a tab character, a vertical tab character, or a form-feed character. A delay is useful in cases where a slow output device such as a teleprinter, which requires a delay for carriage returns, is attached to the system. The following table shows the kind of delay implemented by each bit. Any combination of bits is valid although the system ignores bits 6 and 7.

Bit Pattern	Kind of Delay Affected	Length of Delay (msec)
00000001	New-line	10
00000010	New-line	20
00000100	Carriage return	10
00001000	Carriage return	20
00010000	Tab	20
00100000	Vertical tab	240

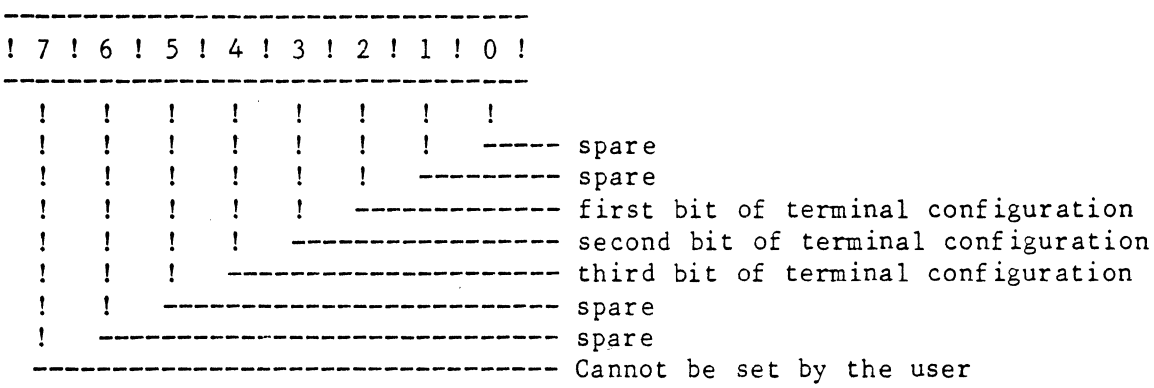
The file "/lib/systty" defines certain combinations of bits in the delay byte as follows:

Bit Pattern	Constant	Kind of Delay	Length of Delay (msec)
00000011	DELNL	New-line	30
00001100	DELCR	Carriage return	30
00010000	DELTB	Tab	20
00100000	DELVT	Vertical tab	240
00100000	DELFF	Form-feed	240

The third and fourth bytes of the buffer define the line-cancel and backspace characters. The default line-cancel character is control-X; the default backspace, control-H.

The fifth byte of the "ttyset" buffer is the terminal speed byte. This byte presently implements only four bits. Bits 2, 3, and 4 define the configuration of the terminal; bit 7 is a flag which, when set, indicates that the terminal has input characters waiting to be consumed by the program. This bit is only meaningful when read--that is, the input-ready condition should not be set with the "ttyset" system call. A picture of this byte follows.

Terminal speed byte (tt_spd):



Under normal input operations the "input ready to be consumed" bit does not come on until an entire line has been input and terminated by a carriage return. There are special input modes which can be established, however, where the "input ready to be consumed" bit will come on as soon as a single character is input. These modes, "raw I/O mode" and "single character input mode" are described in Section 6.2 of the 68xxx UniFLEX Programmer's Guide.

The following table shows the configuration of the terminal for all possible settings of the terminal configuration bits:

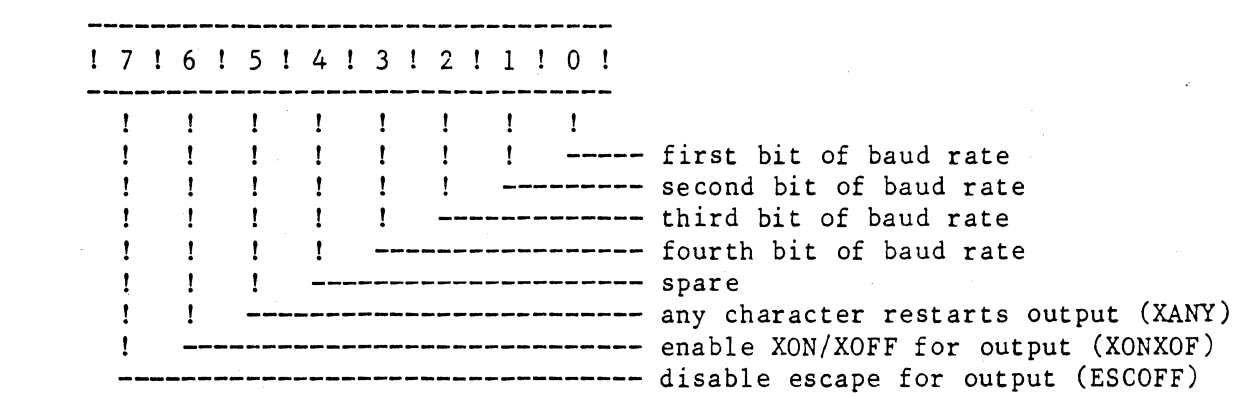
Terminal Configuration (Bit Pattern)	Data Bits	Stop Bits	Parity
0 0 0	7	2	Even
0 0 1	7	2	Odd
0 1 0	7	1	Even
0 1 1	7	1	Odd
1 0 0	8	2	None
1 0 1	8	1	None
1 1 0	8	1	Even
1 1 1	8	1	Odd

(continued)

The last byte in the "ttyset" buffer is the stop output byte, which defines the baud rate and which characters may be used to stop and start output. A user may stop and start output to a terminal by one of two methods: using the escape key or using XON/XOFF processing.

The escape key method permits a user to type an escape character (hexadecimal 1B) to stop output. A subsequent escape character restarts the output. The XON/XOFF method permits a user to type an XOFF character (hexadecimal 13) to stop output and a subsequent XON character (hexadecimal 11) to restart it. Many terminals produce XON and XOFF characters automatically to prevent the computer from sending too many characters to the terminal at once. The escape and XON/XOFF mechanisms can be independently enabled or disabled by setting or clearing the appropriate bits in the byte "tt_spr". A picture of the byte follows. The file "/lib/systty" defines the constants whose names are shown in parentheses.

Terminal output byte (tt_spr):



When XANY is in effect, the terminal drivers restart output stopped by either an escape or an XOFF character when the user types any character.

The following table shows the baud rate defined by all possible settings of the first four bits of the stop output byte:

Bit Pattern	Baud Rate	Bit Pattern	Baud Rate
0 0 0 0	--	1 0 0 0	1200
0 0 0 1	75	1 0 0 1	1800
0 0 1 0	110	1 0 1 0	2400
0 0 1 1	134.5	1 0 1 1	3600
0 1 0 0	150	1 1 0 0	4800
0 1 0 1	200	1 1 0 1	7200
0 1 1 0	300	1 1 1 0	9600
0 1 1 1	600	1 1 1 1	19200

(continued)

Not all hardware supports all of these baud rates, and not all hardware allows the dynamic changing of baud rates.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file.

EBARG
An argument to the system call is invalid.

ENTTY
The file referenced by <file_des> is not a character device.

SEE ALSO

ttyget
68xxx UniFLEX Programmer's Guide

unlink

Remove a link to a file.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys unlink,<file_name>
```

Arguments

<file_name>	The address of the null-terminated name to unlink.

DESCRIPTION

The "unlink" system call removes the entry specified by <file_name> from its parent directory. If that entry was the only one on the system that referenced the file (i.e., the link count is 0 after the link is removed) and the file is closed, the operating system deletes the file. If the system call removes the last link to an open file, the operating system postpones deleting the file until it is closed.

The current effective user must have write permission in the directory containing the specified file.

ERRORS REPORTED

EMSDR
The path to <file_name> cannot be followed.

ENDR
A part of the path to <file_name> is not a directory.

ENOFL
No file on the system corresponds to the specified name.

EPRM
The directory containing the specified file does not grant write permission to the current effective user.

SEE ALSO

link
Commands: diskrepair

unmnt

Unmount the medium in a device.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys unmnt,<dev_name>
```

Arguments

<dev_name>	The address of the null-terminated name of the device containing the medium to unmount. The specified device must be a block device.

DESCRIPTION

The "unmnt" system call unmounts the medium in the specified block device from a node of the directory tree. If the indicator that the medium is mounted for reading and writing is set, the "unmnt" system call clears it.

Only the system manager may invoke this system call.

NOTES

- A medium that was mounted for reading and writing but was not unmounted correctly cannot be mounted again until it has been repaired by the "diskrepair" command.

ERRORS REPORTED

EBDEV
The argument <dev_name> does not reference a device.

EBSY
A file on the medium in the specified device is currently open or a task has a directory on the medium as its working directory.

EMSDR
The path to <file_name> cannot be followed.

ENMNT
The medium in the specified device is not mounted.

ENOFL
No file on the system corresponds to the specified name.

EPRM
Only the system manager may invoke this system call.

SEE ALSO

mount
Commands: diskrepair

update

Update all disks on the system.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys update
```

DESCRIPTION

The "update" system call writes all data in memory that are destined for a disk to the appropriate disk.

urec

Remove an entry from the operating system's lock table.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

```
sys urec
```

Arguments

<file_des> The file descriptor for the file containing the record to unlock.

DESCRIPTION

The "urec" system call removes from the operating system's lock table the current task's entry for the open file referenced by <file_des>.

The operating system removes all entries a task makes in the lock table when the task terminates.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file, or it references a pipe, a character device, a block device, or a pseudoterminal.

EBARG
An argument to the system call is invalid.

SEE ALSO

lrec

vfork

Create a new task.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys vfork
```

Returns

```
To parent task: <child_task's_ID> in D0
To child task:  0 in D0
```

Arguments

`<child_task's_ID>` The task ID assigned to the child task.

DESCRIPTION

The "vfork" system call creates a new task (the child task) that is a copy of the current task (the parent task). This system call, which is available only on systems with virtual memory, is more efficient than the "fork" system call because the child task shares the parent task's user-accessible memory. After invoking "vfork", the parent task sleeps until the child task either terminates or invokes the "exec" or "exece" system call.

The child task has the same priority, user ID, effective user-ID, controlling terminal information, default permissions-mask, working directory, signal handling set-up, profiling information, and user-accessible memory as the parent task. However, it differs from the parent task in the following ways: its task ID is different; its parent task-ID is the task ID of the parent task; its file descriptors are the same, but they are located in a different place in memory; its system and user CPU times are set to 0. The child task may neither change the size of its memory nor invoke the "memman", "fork", or "vfork" system call.

The operating system prevents the parent task from executing until the child task executes an "exec" or an "exece" system call or until the child task terminates. The parent task then resumes execution 2 bytes after the "vfork" call. Obviously, then, the first instruction in the new task must be a short branch (requiring only 2 bytes). Each task determines where to resume by looking at the contents of the D0 register immediately after execution of the "vfork" call.

NOTES

- The user should make sure that the child task does not alter the stack frame in any way or change data that the parent does not expect to change. . If invoked on a system that does not have virtual memory, the "vfork" system call behaves like the "fork" system call.

ERRORS REPORTED

ETMTS
: Either the maximum number of tasks allowed to a user or the maximum number of tasks allowed to the operating system has been reached. The system manager can alter either or both of these limits with the "tune" command up to the system-dependent maxima.

EVFORK
: The current task shares its memory with its parent and may not invoke this system call.

SEE ALSO

fork

wait

Suspend the task until a child task terminates.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys wait
```

Returns

<task_ID> in D0
<term_status> in A0

Arguments

<task_ID>	The task ID of the terminated child task.
<term_status>	The termination status of the terminated child task. If the "term" system call terminated the task, the upper byte of the word is 0, and the lower byte has the value that was in the D0 register when "term" was invoked (i.e., the termination status set by "term"). If the system terminated the child task abnormally, the upper byte is nonzero. The low-order 7 bits of this byte contain the number of the signal that terminated the task. The high-order bit is 1 if the system produced a core dump; 0, if it did not. The value of the lower byte is unpredictable.

DESCRIPTION

The "wait" system call suspends the task until a child task terminates. If more than one child task has terminated at the time the task invokes the system call, the user cannot determine ahead of time to which task the values returned refer.

ERRORS REPORTED

ENCHD
No child tasks are active.

EINTR
The task caught an interrupt which caused this system call to terminate abnormally.

SEE ALSO

cpint

write

Write data to an open file.

ASSEMBLY LANGUAGE SYNTAX

Expected

<file_des> in D0

Syntax Statement

sys write,<buf_add>,<count>

Returns

<bytes_written>

Arguments

<file_des>	The address of the null-terminated name of the file to which to write.
<buf_add>	The address of the buffer containing the data to write.
<count>	The number of bytes of data to write from the buffer to the file.
<bytes_written>	The number of bytes of data actually written to the file.

DESCRIPTION

The "write" system call writes data from the buffer at <buf_add> to the file referenced by <file_des>. It writes at most <count> bytes of data. The system call may write less data than requested if it is writing to a slow device, such as a terminal, and the task catches an interrupt.

The "write" system call is most efficient when both <buf_add> and <count> are multiples of 512.

ERRORS REPORTED

EBADF
The file descriptor does not reference an open file.

EBARG
An argument to the system call is invalid.

EDFUL
The device containing the specified file is full.

EINTR
The task caught an interrupt, which caused the system call to end abnormally.

EIO
The operating system returned an I/O error.

EPIPE
The system call attempted to write to a broken pipe (a pipe whose file descriptor for reading is closed.

SEE ALSO

read

yield_CPU

Yield the CPU to another task of equal priority.

ASSEMBLY LANGUAGE SYNTAX

Syntax Statement

```
sys yield_CPU
```

DESCRIPTION

The "yield_CPU" system call yields the central processing unit (CPU) to another task of equal priority if such a task is waiting to execute. The ability to yield the CPU is especially important for real-time tasks but may also be of some use to non-real-time tasks.

Normally, the operating system lowers the priority of a non-real-time task as time passes so that all tasks can have more or less equal access to the system's resources. The system does, however, increase the priority of a task that has been suspended. For instance, if a task requests information from a file and the operating system must fetch that information from a disk, the system suspends the task until the information is transferred from the disk to memory. Once the information is available, the operating system wakes the task and increases its priority. A task that performs many I/O operations may take over the system as its priority is repeatedly increased unless it deliberately surrenders the CPU with the "yield_CPU" system call.

Real-time tasks have fixed priorities, and any real-time task has a higher priority than any non-real-time task. Therefore, unless a real-time task requests a service from the operating system which causes the system to put the task to sleep, the task can continue to execute until a real-time task of higher priority usurps the CPU. The "yield_CPU" system call allows two or more real-time tasks with the same priority to share the CPU.

NOTES

- If a task tries to execute when the CPU is in use by a task of equal or higher priority, the operating system places the task at the end of a queue of tasks of equal priority. When a task yields the CPU, the operating system places that task at the end of the queue of tasks of the same priority and executes the first task in that queue. Thus, if no tasks of equal priority are in the queue when the task invokes "yield_CPU", the system call has no obvious effect.

yield_CPU-2

SEE ALSO

make_realtime