CHAPTER 6

Thread Interface Specifications

This chapter discusses the specifics of the kernel's thread interfaces. This includes status functions related to threads. Properties associated with threads, such as special ports, are included here as well. Functions that apply to more than one thread appear in the task interface chapter.

catch_exception_raise

catch_exception_raise Server Interface — Handles the occurrence of an exception within a thread

SYNOPSIS

kern_return_t **catch_exception_raise**(mach_port_t *exception_port,*
mach_port_t *thread,* mach_port_t *task,* int
exception, int *code,* int *subcode)*;

DESCRIPTION

A **catch_exception_raise** function is called as the result of a kernel message indicating that an exception occurred within a thread. *exception_port* is the port named via **thread_set_special_port** or **task_set_special_port** as the port that responds when the thread takes an exception.

PARAMETERS

exception_port
The port to which the exception notification was sent.

thread
The control port to the thread taking the exception.

task
The control port to the task containing the thread taking the exception.

exception
The type of the exception, as defined in **<mach/exception.h>**. The machine independent values raised by all implementations are:

EXC_BAD_ACCESS
Could not access memory. *code* contains **kern_return_t** describing error. *subcode* contains bad memory address.

EXC_BAD_INSTRUCTION
Instruction failed. Illegal or undefined instruction or operand

EXC_ARITHMETIC
Arithmetic exception; exact nature of exception is in *code* field

EXC_EMULATION
Emulation instruction. Emulation support instruction encountered. Details in *code* and *subcode* fields.

EXC_SOFTWARE
Software generated exception; exact exception is in *code* field. Codes 0 - 0xFFFF reserved to hardware; codes 0x10000 - 0x1FFFF reserved for OS emulation (Unix).

EXC_BREAKPOINT
Trace, breakpoint, etc. Details in *code* field.

code
A code indicating a particular instance of *exception*.

subcode
A specific type of *code*.

NOTES

When an exception occurs in a thread, the thread sends an exception message to its exception port, blocking in the kernel waiting for the receipt of a reply. It is assumed that some task is listening (most likely with **mach_msg_server**) to this port, using the **exc_server** function to decode the messages and then call the linked in **catch_exception_raise**. It is the job of **catch_exception_raise** to handle the exception and decide the course of action for *thread*. The state of the blocked thread can be examined with **thread_get_state**.

If the thread should continue from the point of exception, **catch_exception_raise** would return KERN_SUCCESS. This causes a reply message to be sent to the kernel, which will allow the thread to continue from the point of the exception.

If some other action should be taken by *thread*, the following actions should be performed by **catch_exception_raise**:

- **thread_suspend**. This keeps the thread from proceeding after the next step.
- **thread_abort**. This aborts the message receive operation currently blocking the thread.
- **thread_set_state**. Set the thread's state so that it continues doing something else.
- **thread_resume**. Let the thread start running from its new state.
- Return a value other than KERN_SUCCESS so that no reply message is sent. (Actually, the kernel uses a send once right to send the exception message, which **thread_abort** destroys, so replying to the message is harmless.)

The thread can always be destroyed with **thread_terminate**.

A thread can have two exception ports active for it: its thread exception port and the task exception port. If an exception message is sent to the thread exception port (if it exists), and a reply message contains a return value other than KERN_SUCCESS, the kernel will then send the exception message to the task exception port. If that exception message receives a reply message with other than a return value of KERN_SUCCESS, the thread is terminated. Note that this behavior cannot be obtained by using the **catch_exception_-raise** interface called by **exc_server** and **mach_msg_server**, since those functions will either return a reply message with a KERN_SUCCESS value, or none at all.

RETURN VALUE

KERN_SUCCESS
: The thread is to continue from the point of exception.

Other values indicate that the exception was handled directly and the thread was restarted or terminated by the exception handler.

RELATED INFORMATION

Functions: **exc_server**, **thread_abort**, **task_get_special_port**, **thread_-get_special_port**, **thread_get_state**, **thread_resume**, **task_set_special_-port,** **thread_set_special_port**, **thread_set_state**, **thread_suspend**, **thread_terminate.**

exc_server

exc_server Function — Handles kernel messages for an exception handler

LIBRARY

libmach.a

SYNOPSIS

boolean_t **exc_server**(mach_msg_header_t **in_msg,* mach_msg_header_t
out_msg);

DESCRIPTION

The **exc_server** function is the MIG generated server handling function to handle messages from the kernel relating to the occurrence of an exception in a thread. Such messages are delivered to the exception port set via **thread_-set_special_port** or **task_set_special_port**. When an exception occurs in a thread, the thread sends an exception message to its exception port, blocking in the kernel waiting for the receipt of a reply. The **exc_server** function performs all necessary argument handling for this kernel message and calls **catch_exception_raise**, which should handle the exception. If **catch_exception_raise** returns KERN_SUCCESS, a reply message will be sent, allowing the thread to continue from the point of the exception; otherwise, no reply message is sent and **catch_exception_raise** must have dealt with the exception thread directly.

PARAMETERS

in_msg
: The exception message received from the kernel.

out_msg
: A reply message.

RETURN VALUE

TRUE
: The message was handled and the appropriate function was called.

FALSE
: The message did not apply to the exception mechanism and no other action was taken.

RELATED INFORMATION

Functions: **thread_set_special_port**, **task_set_special_port**, **catch_exception_raise**.

mach_thread_self

mach_thread_self Function — Returns the thread self port

LIBRARY

libmach.a

#include <mach/mach_traps.h>

SYNOPSIS

mach_port_t **mach_thread_self**();

DESCRIPTION

The **mach_thread_self** function returns send rights to the thread's own port.

PARAMETERS

None

NOTES

This function is actually a system call.

RETURN VALUE

Send rights to the thread's port.

RELATED INFORMATION

Functions: **thread_info**.

swtch

swtch Function — Attempt a context switch

LIBRARY

libmach.a

SYNOPSIS

boolean_t **swtch**();

DESCRIPTION

The **swtch** function attempts to context switch the current thread off the processor.

This function is useful in user level lock management routines. If the current thread cannot make progress because of some lock, it would execute the **swtch** function. When this returns, the thread should once again try to make progress by attempting to obtain its lock.

This function returns a flag indicating whether there is anything else for the processor to do. If there is nothing else, the thread can spin waiting for its lock, instead of continuing to call **swtch**.

PARAMETERS

None

NOTES

This function is actually a system call.

RETURN VALUE

TRUE
: There are other threads that the processor could run.

FALSE
: The processor has nothing better to do.

RELATED INFORMATION

Functions: **swtch_pri**, **thread_abort**, **thread_switch**.

swtch_pri

swtch_pri Function — Attempt a context switch to low priority

LIBRARY

libmach.a

SYNOPSIS

boolean_t **swtch_pri** (int *priority*);

DESCRIPTION

The **swtch_pri** function attempts to context switch the current thread off the processor. The thread's priority is lowered to the minimum possible value during this time. The priority of the thread will be restored when it is awakened.

This function is useful in user level lock management routines. If the current thread cannot make progress because of some lock, it would execute the **swtch_pri** function. When this returns, the thread should once again try to make progress by attempting to obtain its lock.

This function returns a flag indicating whether there is anything else for the processor to do. If there is nothing else, the thread can spin waiting for its lock, instead of continuing to call **swtch_pri**.

PARAMETERS

priority
: Currently not used.

NOTES

This function is actually a system call.

RETURN VALUE

TRUE
: There are other threads that the processor could run.

FALSE
: The processor has nothing better to do.

RELATED INFORMATION

Functions: **swtch**, **thread_abort**, **thread_switch**.

thread_abort

thread_abort Function — Aborts a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_abort** (thread_t *target_thread)*;

DESCRIPTION

The **thread_abort** function aborts page faults and any message primitive calls (**mach_msg**, **mach_msg_receive**, and **mach_msg_send**) in use by *target_thread*. (Note, though, that the message calls retry interrupted message operations unless MACH_SEND_INTERRUPT and MACH_RCV_INTERRUPT are specified.) Priority depressions are also aborted. The call returns a code indicating that it was interrupted. The call is interrupted even if the thread (or the task containing it) is suspended. If it is suspended, the thread receives the interrupt when it resumes.

If its state is not modified before it resumes, the thread will retry an aborted page fault. The message primitives return appropriate values, as follows:

- **mach_msg_receive** returns MACH_RCV_INTERRUPTED.
- **mach_msg** returns either MACH_SEND_INTERRUPTED or MACH_RCV_INTERRUPTED, depending on whether the send or the receive side was interrupted.
- **mach_msg_send** returns MACH_SEND_INTERRUPTED.

The basic purpose of **thread_abort** is to let one thread cleanly stop another thread (*target_thread*). The target thread is stopped in such a manner that its future execution can be controlled in a predictable way.

PARAMETERS

target_thread
The thread to be aborted.

NOTES

By way of comparison, the **thread_suspend** function keeps the target thread from executing any further instructions at the user level, including the return from a system call. The **thread_get_state** function returns the thread's user state, while **thread_set_state** allows modification of the user state.

A problem occurs if a suspended thread had been executing within a system call. In this case, the thread has, not only a user state, but an associated kernel state. (The kernel state cannot be changed with **thread_set_state**.) As a result, when the thread resumes, the system call can return, producing a change in the user state and, possibly, user memory.

For a thread executing within a system call, **thread_abort** aborts the kernel call from the thread's point of view. Specifically, it resets the kernel state so that the thread will resume execution at the system call return, with the return code value set to one of the interrupted codes. The system call itself is either completed entirely or aborted entirely, depending on when the abort is received. As a result, if the thread's user state has been modified by **thread_set_state**, it will not be altered un-predictably by any unexpected system call side effects.

For example, to simulate a POSIX signal, use the following sequence of calls:

thread_suspend — To stop the thread.

thread_abort — To interrupt any system call in progress and set the return value to "interrupted". Because the thread is already stopped, it will not return to user code.

thread_set_state — To modify the thread's user state to simulate a procedure call to the signal handler.

thread_resume — To resume execution at the signal handler. If the thread's stack is set up correctly, the thread can return to the interrupted system call. Note that the code to push an extra stack frame and change the registers is highly machine dependent.

CAUTIONS

As a rule, do not use **thread_abort** on a non-suspended thread. This operation is very risky because it is difficult to know which system trap, if any, is executing and whether an interrupt return will result in some useful action by the thread.

RETURN VALUE

KERN_SUCCESS
: The thread received an interrupt.

KERN_INVALID_ARGUMENT
: *target_thread* is not a valid thread.

RELATED INFORMATION

Functions: **thread_get_state**, **thread_info**, **thread_set_state**, **thread_suspend**, **thread_terminate**.

thread_create

thread_create Function — Creates a thread within a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_create** (task_t *parent_task,* thread_t **child_thread)*;

DESCRIPTION

The **thread_create** function creates a new thread within *parent_task*. The new thread has a suspend count of one and no processor state.

The new thread holds a send right for its thread kernel port. A send right for the thread's kernel port is also returned to the calling task or thread in *child_thread*. The new thread's exception port is set to MACH_PORT_NULL.

PARAMETERS

parent_task
: The task that is to contain the new thread.

child_thread
: [out] The kernel-assigned name for the new thread.

NOTES

To get a new thread running, first use **thread_set_state** to set a processor state for the thread. Then, use **thread_resume** to schedule the thread for execution.

RETURN VALUE

KERN_SUCCESS
: A new thread has been created.

KERN_INVALID_ARGUMENT
: *parent_task* is not a valid task port.

KERN_RESOURCE_SHORTAGE
: Some critical kernel resource is unavailable.

RELATED INFORMATION

Functions: **task_create**, **task_threads**, **thread_get_special_port**, **thread_-get_state**, **thread_resume**, **thread_set_special_port**, **thread_set_state**, **thread_suspend**, **thread_terminate**.

thread_get_exception_port

thread_get_exception_port Macro — Returns a send right to the exception port

LIBRARY

libmach.a

#include <mach.h> (**<mach/thread_special_ports.h>**)

SYNOPSIS

kern_return_t **thread_get_exception_port**(thread_t *thread,* mach_port_t *special_port)*;

DESCRIPTION

The **thread_get_exception_port** function returns a send right for the exception port belonging to *thread*. This is a short-hand for **thread_get_special_port** (*thread*, THREAD_EXCEPTION_PORT, *special_port*).

PARAMETERS

thread
The thread for which to return the port's send right.

special_port
[out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
The port was returned.

KERN_INVALID_ARGUMENT
thread is not a valid thread.

RELATED INFORMATION

Functions: **thread_get_special_port**.

thread_get_kernel_port

thread_get_kernel_port Macro — Returns a send right to the kernel port

LIBRARY

libmach.a

#include <mach.h> (**<mach/thread_special_ports.h>**)

SYNOPSIS

kern_return_t **thread_get_kernel_port**(thread_t *thread,* mach_port_t **special_port)*;

DESCRIPTION

The **thread_get_kernel_port** function returns a send right for the kernel port belonging to *thread*. This is a short-hand for **thread_get_special_port** (*thread*, THREAD_KERNEL_PORT, *special_port*).

PARAMETERS

thread
The thread for which to return the port's send right.

special_port
[out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
The port was returned.

KERN_INVALID_ARGUMENT
thread is not a valid thread.

RELATED INFORMATION

Functions: **thread_get_special_port**.

thread_get_special_port

thread_get_special_port Function — Returns a send right to a special port

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_get_special_port**(thread_t *thread,* int *which_port,*
mach_port_t **special_port)*;

DESCRIPTION

The **thread_get_special_port** function returns a send right for a special port belonging to *thread*.

The thread kernel port is a port for which the kernel holds the receive right. The kernel uses this port to identify the thread.

If one thread has a send right for the kernel port of another thread, it can use the port to perform kernel operations for the other thread. Send rights for a kernel port normally are held only by the thread to which the port belongs, or by the task that contains the thread. Using the **mach_msg** function, however, any thread can pass a send right for its kernel port to another thread.

PARAMETERS

thread
The thread for which to return the port's send right.

which_port
The special port for which the send right is requested. Valid values are:

THREAD_EXCEPTION_PORT
The thread's exception port. Used to receive exception messages from the kernel.

THREAD_KERNEL_PORT
The port used to name the thread. Used to invoke operations that affect the thread.

special_port
[out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
The port was returned.

KERN_INVALID_ARGUMENT
thread is not a valid thread or *which_port* is not a valid port selector.

RELATED INFORMATION

Functions: **mach_thread_self**, **task_get_special_port**, **task_set_special_-port**, **thread_create**, **thread_set_special_port**.

thread_get_state

thread_get_state Function — Returns the execution state for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_get_state**(thread_t *target_thread,* int *flavor,*
thread_get_state_data_t *old_state,*
mach_msg_type_number_t **old_stateCnt)*;

DESCRIPTION

The **thread_get_state** function returns the execution state (for example, the machine registers) for *target_thread*. *flavor* specifies the type of state information returned.

For *old_state*, the calling thread supplies an array of integers. On return, *old_-state* contains the requested information.

For *old_stateCnt*, the calling thread specifies the maximum number of integers in *old_state*. On return, *old_stateCnt* contains the actual number of integers in *old_state*.

PARAMETERS

target_thread
The thread for which the execution state is to be returned. The calling thread cannot specify itself.

flavor
The type of execution state to be returned. Valid values correspond to supported machined architectures.

old_state
[in/out array] Array of state information for the specified thread.

old_stateCnt
[in/out] The size of the state array. The maximum size is defined by THREAD_STATE_MAX.

RETURN VALUE

KERN_SUCCESS
The state has been returned.

KERN_INVALID_ARGUMENT
: *target_thread* is not a valid thread, or specifies the calling thread, or *flavor* is not a valid type.

MIG_ARRAY_TOO_LARGE
: The returned array is too large for *old_state*. The function fills *old_-state* and sets *old_stateCnt* to the number of elements that would have been returned if there had been enough space.

RELATED INFORMATION

Functions: **task_info**, **thread_info**, **thread_set_state**.

thread_info

thread_info Function — Returns information about a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_info** (thread_t *target_thread,* int *flavor,* thread_info_t *thread_info,* mach_msg_type_number_t **thread_infoCnt)*;

DESCRIPTION

The **thread_info** function returns an information array of type *flavor.*

For *thread_info*, the calling thread supplies an array of integers. On return, *thread_info* contains the requested information.

For *thread_infoCnt*, the calling thread specifies the maximum number of integers in *thread_info*. On return, *thread_infoCnt* contains the actual number of integers in *thread_info*.

Currently, THREAD_BASIC_INFO and THREAD_SCHED_INFO are the only types of information supported. The size is defined by THREAD_BASIC_INFO_COUNT or THREAD_SCHED_INFO_COUNT, respectively.

PARAMETERS

target_thread
The thread for which the information is to be returned.

flavor
The type of information to be returned. Valid values are:

THREAD_BASIC_INFO
Returns basic information about the thread, such as the thread's run state and suspend count.

THREAD_SCHED_INFO
Returns scheduling information about the thread, such as priority and scheduling policy.

thread_info
[in/out] Information about the specified thread.

thread_infoCnt
[in/out] The size of the information structure. The maximum size is defined by THREAD_INFO_MAX. Possible values are THREAD_-BASIC_INFO_COUNT (for THREAD_BASIC_INFO) and THREAD_SCHED_INFO_COUNT (for THREAD_SCHED_INFO).

RETURN VALUE

KERN_SUCCESS
The thread information has been returned.

KERN_INVALID_ARGUMENT
target_thread is not a valid thread or *flavor* is not a valid type.

MIG_ARRAY_TOO_LARGE
The returned array is too large for *thread_info*. The function fills *thread_info* and sets *thread_infoCnt* to the number of elements that would have been returned if there had been enough space.

RELATED INFORMATION

Functions: **task_info**, **task_threads**, **thread_get_special_port**, **thread_get_-state**, **thread_set_special_port**, **thread_set_state**.

Data Structures: **thread_basic_info**, **thread_sched_info**.

thread_max_priority

thread_max_priority Function — Sets the maximum scheduling priority for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_max_priority**(thread_t *thread,* processor_set_t *processor_set,* int *priority)*;

DESCRIPTION

The **thread_max_priority** function sets the maximum scheduling priority for *thread*.

Threads have three priorities associated with them by the system:

- A priority value which can be set by the thread to any value up to a maximum priority. Newly created threads obtain their priority from their task.
- A maximum priority value which can be raised only via privileged operation so that users may not unfairly compete with other users in their processor set. Newly created threads obtain their maximum priority from that of their assigned processor set.
- A scheduled priority value which is used to make scheduling decisions for the thread. This value is determined on the basis of the user priority value by the scheduling policy (for timesharing, this means adding an increment derived from CPU usage).

This function changes the maximum priority for the thread. Because this function requires the presentation of the corresponding processor set control port, this call can reset the maximum priority to any legal value.

PARAMETERS

thread
The thread whose maximum scheduling priority is to be set.

processor_set
The control port for the processor set to which the thread is currently assigned.

priority
The new maximum priority for the thread.

RETURN VALUE

KERN_SUCCESS
: The priority has been set.

KERN_INVALID_ARGUMENT
: *thread* is not a valid thread, or *processor_set* does not name the processor set to which *thread* is currently assigned.

RELATED INFORMATION

Functions: **thread_priority, thread_policy, task_priority, processor_set_-max_priority**.

thread_policy

thread_policy Function — Sets the scheduling policy to apply to a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_policy** (thread_t *thread,* int *policy,* int *data)*;

DESCRIPTION

The **thread_policy** function sets the scheduling policy to be applied to *thread*.

PARAMETERS

thread
: The thread scheduling policy is to be set.

policy
: Policy to be set. The values currently defined are POLICY_TIMESHARE and POLICY_FIXEDPRI.

data
: Policy specific data. Currently, this value is used only for POLICY_FIXEDPRI, in which case it is the quantum to be used (in milliseconds); to be meaningful, this value must be a multiple of the basic system quantum (which can be obtained from **host_info**).

RETURN VALUE

KERN_SUCCESS
: The policy has been set.

KERN_INVALID_ARGUMENT
: *thread* is not a valid thread, or *policy* is not a recognized scheduling policy value.

KERN_FAILURE
: The processor set to which *thread* is currently assigned does not permit *policy.*

RELATED INFORMATION

Functions: **processor_set_policy_enable, processor_set_policy_disable**.

thread_priority

thread_priority Function — Sets the scheduling priority for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_priority** (thread_t *thread,* int *priority,* boolean_t *set_max)*;

DESCRIPTION

The **thread_priority** function sets the scheduling priority for *thread*.

PARAMETERS

thread
The thread whose scheduling priority is to be set.

priority
The new priority for the thread.

set_max
True if the thread's maximum priority should also be set.

NOTES

Threads have three priorities associated with them by the system:

- A priority value which can be set by the thread to any value up to a maximum priority. Newly created threads obtain their priority from their task.
- A maximum priority value which can be raised only via privileged operation so that users may not unfairly compete with other users in their processor set. Newly created threads obtain their maximum priority from that of their assigned processor set.
- A scheduled priority value which is sued to make scheduling decisions for the thread. This value is determined on the basis of the user priority value by the scheduling policy (for timesharing, this means adding an increment derived from CPU usage).

This function changes the priority and optionally the maximum priority (if *set_max* is TRUE) for *thread*. Priorities range from 0 to 31, where lower numbers denote higher priorities. If the new priority is higher than the priority of the current thread, preemption may occur as a result of this call. This call will fail if *priority* is greater than the current maximum priority of the thread. As a result, this call can only lower the value of a thread's maximum priority.

RETURN VALUE

KERN_SUCCESS
: The priority has been set.

KERN_INVALID_ARGUMENT
: *thread* is not a valid thread, or the priority value is out of range for priority values.

KERN_FAILURE
: The requested operation would violate the thread's maximum priority.

RELATED INFORMATION

Functions: **thread_max_priority, thread_policy, task_priority, processor_set_max_priority**.

thread_resume

thread_resume Function — Resumes a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_resume** (thread_t *target_thread)*;

DESCRIPTION

The **thread_resume** function decrements the suspend count for *target_thread* by one. The thread is resumed if its suspend count goes to zero. If the suspend count is still positive, you must repeat **thread_resume** until the count reaches zero.

PARAMETERS

target_thread
The thread to be resumed.

RETURN VALUE

KERN_SUCCESS
The thread's suspend count has been decremented.

KERN_FAILURE
The thread's suspend count is already at zero. A suspend count must be either zero or positive.

KERN_INVALID_ARGUMENT
target_thread is not a valid thread.

RELATED INFORMATION

Functions: **task_resume**, **task_suspend**, **thread_create**, **thread_info**, **thread_suspend**, **thread_terminate**.

thread_set_exception_port

thread_set_exception_port Macro — Sets the exception port for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/thread_special_ports.h>**)

SYNOPSIS

kern_return_t **thread_set_exception_port**(thread_t *thread,* mach_port_t *special_port)*;

DESCRIPTION

The **thread_set_exception_port** macro sets the exception port belonging to *thread*. This is a short-hand for **thread_set_special_port** (*thread*, THREAD_-EXCEPTION_PORT, *special_port*).

PARAMETERS

thread
: The thread for which to set the port.

special_port
: The value for the port.

RETURN VALUE

KERN_SUCCESS
: The port was set.

KERN_INVALID_ARGUMENT
: *thread* is not a valid thread.

RELATED INFORMATION

Functions: **thread_set_special_port**.

thread_set_kernel_port

thread_set_kernel_port Macro — Sets the kernel port for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/thread_special_ports.h>**)

SYNOPSIS

kern_return_t **thread_set_kernel_port**(thread_t *thread,* mach_port_t *special_port)*;

DESCRIPTION

The **thread_set_kernel_port** macro sets the exception port belonging to *thread*. This is a short-hand for **thread_set_special_port** (*thread*, THREAD_-KERNEL_PORT, *special_port*).

PARAMETERS

thread
The thread for which to set the port.

special_port
The value for the port.

RETURN VALUE

KERN_SUCCESS
The port was set.

KERN_INVALID_ARGUMENT
thread is not a valid thread.

RELATED INFORMATION

Functions: **thread_set_special_port**.

thread_set_special_port

thread_set_special_port Function — Sets a special port for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_set_special_port**(thread_t *thread,* int *which_port,*
mach_port_t *special_port)*;

DESCRIPTION

The **thread_set_special_port** function sets a special port belonging to *thread*.

PARAMETERS

thread
The thread for which to set the port.

which_port
The special port to be set. Valid values are:

THREAD_EXCEPTION_PORT
The thread's exception port. Used to receive exception messages from the kernel.

THREAD_KERNEL_PORT
The thread's kernel port. Used by the kernel to receive messages from the thread.

special_port
The value for the port.

RETURN VALUE

KERN_SUCCESS
The port was set.

KERN_INVALID_ARGUMENT
thread is not a valid thread or *which_port* is not a valid port selector.

RELATED INFORMATION

Functions: **mach_thread_self**, **task_get_special_port**, **task_set_special_port**, **thread_create**, **thread_get_special_port**.

thread_set_state

thread_set_state Function — Sets the execution state for a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_set_state**(thread_t *target_thread,* int *flavor,*
thread_state_data_t *new_state,*
mach_msg_type_number_t *new_stateCnt)*;

DESCRIPTION

The **thread_set_state** function sets the execution state (for example, the machine registers) for *target_thread*. *flavor* specifies the type of state to set.

For *new_state*, the calling thread supplies an array of integers.

For *new_stateCnt*, the calling thread specifies the maximum number of integers in *new_state*.

PARAMETERS

target_thread
: The thread for which to set the execution state. The calling thread cannot specify itself.

flavor
: The type of state to set. Valid values correspond to supported machine architecture features.

new_state
: Array of state information for the specified thread.

new_stateCnt
: The size of the state array. The maximum size is defined by THREAD_STATE_MAX.

RETURN VALUE

KERN_SUCCESS
: The state has been set.

KERN_INVALID_ARGUMENT
: *target_thread* is not a valid thread, or specifies the calling thread, or *flavor* is not a valid type.

MIG_ARRAY_TOO_LARGE
: The state array is too large for *new_state*. The function fills *new_-state* and sets *new_stateCnt* to the number of elements that would have been returned if there had been enough space.

RELATED INFORMATION

Functions: **task_info**, **thread_get_state**, **thread_info**.

thread_suspend

thread_suspend Function — Suspends a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_suspend** (thread_t *target_thread)*;

DESCRIPTION

The **thread_suspend** function increments the suspend count for *target_thread* and prevents the thread from executing any more user-level instructions.

In this context, a user-level instruction can be either a machine instruction executed in user mode or a system trap instruction, including a page fault. If a thread is currently executing within a system trap, the kernel code may continue to execute until it reaches the system return code or it may suspend within the kernel code. In either case, the system trap returns when the thread resumes.

To resume a suspended thread, use **thread_resume**. If the suspend count is greater than one, you must issue **thread_resume** that number of times.

PARAMETERS

target_thread
: The thread to be suspended.

CAUTIONS

Unpredictable results may occur if you suspend a thread and alter its user state so that its direction is changed upon resuming. Note that the **thread_abort** function lets you abort a system call only if it is progressing in a predictable way.

RETURN VALUE

KERN_SUCCESS
: The thread has been suspended.

KERN_INVALID_ARGUMENT
: *target_thread* is not a valid thread.

RELATED INFORMATION

Functions: **task_resume**, **task_suspend**, **thread_abort**, **thread_get_state**, **thread_info**, **thread_resume**, **thread_set_state**, **thread_terminate**.

thread_switch

thread_switch Function — Cause context switch with options

LIBRARY

libmach.a

SYNOPSIS

kern_return_t **thread_switch** (thread_t *new_thread,* int *option,* int *time)*;

DESCRIPTION

The **thread_switch** function provides low-level access to the scheduler's context switching code. *new_thread* is a hint that implements hand-off scheduling. The operating system will attempt to switch directly to the new thread (bypassing the normal logic that selects the next thread to run) if possible. Since this is a hint, it may be incorrect; it is ignored if it doesn't specify a thread on the same host as the current thread or if the scheduler cannot switch to that thread (i.e., not runable or already running on another processor). In this case, the normal logic to select the next thread to run is used; the current thread may continue running if there is no other appropriate thread to run.

The *option* argument specifies the interpretation and use of *time*. The possible values (from **<mach/thread_switch.h>**) are:

SWITCH_OPTION_NONE
: The *time* argument is ignored.

SWITCH_OPTION_WAIT
: The thread is blocked for the specified *time*. This wait is cannot be canceled by **thread_resume**; only **thread_abort** can terminate this wait.

SWITCH_OPTION_DEPRESS
: The thread's priority is depressed to the lowest possible value for *time*. The priority depression is aborted when *time* has passed, when the current thread is next run (either via hand-off scheduling or because the processor set has nothing better to do), or when **thread_abort** is applied to the current thread. Changing the thread's priority (via **thread_priority**) will not affect this depression.

The minimum time and units of time can be obtained as the *min_timeout* value from the HOST_SCHED_INFO flavor of **host_info**.

PARAMETERS

new_thread
: Thread to which the processor should switch context.

option
Options applicable to the context switch.

time
Time duration during which the thread should be affected by *option*.

NOTES

thread_switch is often called when the current thread can proceed no further for some reason; the various options and arguments allow information about this reason to be transmitted to the kernel. The *new_thread* argument (hand-off scheduling) is useful when the identity of the thread that must make progress before the current thread runs again is known. The SWITCH_OPTION_WAIT option is used when the amount of time that the current thread must wait before it can do anything useful can be estimated and is fairly short, especially when the identity of the thread for which this thread must wait is not known.

This function is actually a system call.

CAUTIONS

Users should beware of calling **thread_switch** with an invalid hint (e.g., THREAD_NULL) and no option. Because the time-sharing scheduler varies the priority of threads based on usage, this may result in a waste of CPU time if the thread that must be run is of lower priority. The use of the SWITCH_OPTION_DEPRESS option in this situation is highly recommended.

thread_switch ignores policies. Users relying on the preemption semantics of a fixed time policy should be aware that **thread_switch** ignores these semantics; it will run the specified *new_thread* independent of its priority and the priority of any threads that could run instead.

RETURN VALUE

KERN_SUCCESS
The call succeeded.

KERN_INVALID_ARGUMENT
new_thread is not a valid thread, or *option* is not a recognized option.

RELATED INFORMATION

Functions: **swtch**, **swtch_pri**, **thread_abort**.

thread_terminate

thread_terminate Function — Destroys a thread

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **thread_terminate**(thread_t *target_thread)*;

DESCRIPTION

The **thread_terminate** function kills creates *target_thread*.

PARAMETERS

target_thread
The thread to be destroyed.

RETURN VALUE

KERN_SUCCESS
The thread has been killed.

KERN_INVALID_ARGUMENT
target_thread is not a valid thread.

RELATED INFORMATION

Functions: **task_terminate**, **task_threads**, **thread_create**, **thread_resume**, **thread_suspend**.

CHAPTER 7

Task Interface Specifications

This chapter discusses the specifics of the kernel's task interfaces. This includes functions that return status information for a task. Also included are functions that operate upon all or a set of threads within a task.

mach_task_self

mach_task_self Function — Returns the task self port

LIBRARY

libmach.a

#include <mach/mach_traps.h>

SYNOPSIS

mach_port_t **mach_task_self**();

DESCRIPTION

The **mach_task_self** function returns send rights to the task's own port.

The include file **<mach_init.h>** included by **<mach.h>** redefines this function call to simply return the value **mach_task_self_**, cached by the MACH run-time.

PARAMETERS

None

NOTES

This function is actually a system call.

RETURN VALUE

Send rights to the task's port.

RELATED INFORMATION

Functions: **task_info**.

task_create

task_create Function — Creates a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_create** (task_t *parent_task,* boolean_t *inherit_memory,*
task_t **child_task)*;

DESCRIPTION

The **task_create** function creates a new task from *parent_task* and returns the name of the new task in *child_task*. The child task acquires shared or copied parts of the parent's address space (see **vm_inherit**). The child task initially contains no threads.

The child task receives the three following special ports, which are created or copied for it at task creation:

- **task_kernel_port** — The port by which the kernel knows the new child task. The child task holds a send right for this port. The port name is also returned to the calling task.
- **task_bootstrap_port** — The port to which the child task can send a message requesting return of any system service ports that it needs (for example, a port to the Network Name Server or the Environment Manager). The child task inherits a send right for this port from the parent task. The child task can use **task_get_special_port** to change this port.
- **task_exception_port** — A default exception port for the child task, inherited from the parent task. The exception port is the port to which the kernel sends exception messages. Exceptions are synchronous interruptions to the normal flow of program control caused by the program itself. Some exceptions are handled transparently by the kernel, but others must be reported to the program. The child task, or any one of its threads, can change the default exception port to take an active role in exception handling (see **task_get_special_port** or **thread_get_special_port**).

PARAMETERS

parent_task
: The task from which to draw the child task's port rights, resource limits, and address space.

inherit_memory
Address space inheritance indicator. If true, the child task inherits the address space of the parent task. If false, the kernel assigns the child task an empty address space.

child_task
[out] The kernel-assigned name for the new task.

RETURN VALUE

KERN_SUCCESS
A new task has been created.

KERN_INVALID_ARGUMENT
parent_task is not a valid task port.

KERN_RESOURCE_SHORTAGE
Some critical kernel resource is unavailable.

RELATED INFORMATION

Functions: **task_get_special_port**, **task_resume**, **task_set_special_port**, **task_suspend**, **task_terminate**, **task_threads**, **thread_create**, **thread_-resume**, **vm_inherit**.

task_get_bootstrap_port

task_get_bootstrap_port Macro — Returns a send right to the bootstrap port

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_get_bootstrap_port**(task_t *task,* mach_port_t
*special_port);

DESCRIPTION

The **task_get_bootstrap_port** macro returns a send right for the bootstrap port belonging to *task*. This macro is a short-hand for **task_get_special_port** (*task*, TASK_BOOTSTRAP_PORT, *special_port*).

PARAMETERS

task
: The task for which to return the port's send right.

special_port
: [out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
: The port was returned.

KERN_INVALID_ARGUMENT
: *task* is not a valid task.

RELATED INFORMATION

Functions: **task_get_special_port**.

task_get_exception_port

task_get_exception_port Macro — Returns a send right to the exception port

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_get_exception_port**(task_t *task,* mach_port_t *special_port);

DESCRIPTION

The **task_get_exception_port** macro returns a send right for the exception port belonging to *task*. This macro is a short-hand for **task_get_special_port** (*task*, TASK_EXCEPTION_PORT, *special_port*).

PARAMETERS

task
: The task for which to return the port's send right.

special_port
: [out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
: The port was returned.

KERN_INVALID_ARGUMENT
: *task* is not a valid task.

RELATED INFORMATION

Functions: **task_get_special_port**.

task_get_kernel_port

task_get_kernel_port Macro — Returns a send right to the kernel port

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_get_kernel_port**(task_t *task,* mach_port_t **special_port)*;

DESCRIPTION

The **task_get_kernel_port** macro returns a send right for the kernel port belonging to *task*. This macro is a short-hand for **task_get_special_port** (*task*, TASK_KERNEL_PORT, *special_port*).

PARAMETERS

task
The task for which to return the port's send right.

special_port
[out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
The port was returned.

KERN_INVALID_ARGUMENT
task is not a valid task.

RELATED INFORMATION

Functions: **task_get_special_port**.

task_get_special_port

task_get_special_port Function — Returns a send right to a special port

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_get_special_port**(task_t *task,* int *which_port,*
mach_port_t **special_port*);

DESCRIPTION

The **task_get_special_port** function returns a send right for a special port belonging to *task*.

The task kernel port is a port for which the kernel holds the receive right. The kernel uses this port to identify the task.

If one task has a send right for the kernel port of another task, it can use the port to perform kernel operations for the other task. Send rights for a kernel port normally are held only by the task to which the port belongs, or by the task's parent task. Using the **mach_msg** function, however, any task can pass a send right for its kernel port to another task.

PARAMETERS

task
The task for which to return the port's send right.

which_port
The special port for which the send right is requested. Valid values are:

TASK_KERNEL_PORT
The port used to name this task. Used to send messages that affect the task.

TASK_BOOTSTRAP_PORT
The task's bootstrap port. Used to send messages requesting return of other system service ports.

TASK_EXCEPTION_PORT
The task's exception port. Used to receive exception messages from the kernel.

special_port
[out] The returned value for the port.

RETURN VALUE

KERN_SUCCESS
The port was returned.

KERN_INVALID_ARGUMENT
task is not a valid task or *which_port* is not a valid port selector.

RELATED INFORMATION

Functions: **mach_task_self**, **task_create**, **task_set_special_port**, **thread_-get_special_port**, **thread_set_special_port**.

task_info

task_info Function — Returns information about a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_info** (task_t *target_task,* int *flavor,* task_info_t *task_info,*
mach_msg_type_number_t **task_infoCnt)*;

DESCRIPTION

The **task_info** function returns an information array of type *flavor.*

For *task_info*, the calling task or thread supplies an array of integers. On return, *task_info* contains the requested information.

For *task_infoCnt*, the calling task or thread specifies the maximum number of integers in *task_info*. On return, *task_infoCnt* contains the actual number of integers in *task_info*.

Currently, TASK_BASIC_INFO and TASK_THREAD_TIMES_INFO are the only types of information supported. Their sizes are defined by TASK_-BASIC_INFO_COUNT and TASK_THREAD_TIMES_INFO_COUNT, respectively.

PARAMETERS

target_task
The task for which the information is to be returned.

flavor
The type of information to be returned. Valid values are:

TASK_BASIC_INFO
Returns basic information about the task, such as the task's suspend count and number of resident pages.

TASK_THREAD_TIMES_INFO
Returns system and user space run-times for live threads.

task_info
[in/out] Information about the specified task.

task_infoCnt
[in/out] The size of the information structure. The maximum size is defined by TASK_INFO_MAX. Currently, the only valid values are

TASK_BASIC_INFO_COUNT (for TASK_BASIC_INFO) and TASK_THREAD_TIMES_INFO_COUNT (for TASK_THREAD_-TIMES_INFO).

RETURN VALUE

KERN_SUCCESS
The task information has been returned.

KERN_INVALID_ARGUMENT
target_task is not a valid task or *flavor* is not a valid type.

MIG_ARRAY_TOO_LARGE
The returned array is too large for *task_info*. The function fills *task_info* and sets *task_infoCnt* to the number of elements that would have been returned if there had been enough space.

RELATED INFORMATION

Functions: **task_get_special_port**, **task_set_special_port**, **task_threads**, **thread_info**, **thread_get_state**, **thread_set_state**.

Data Structures: **task_basic_info**, **task_thread_times_info**.

task_priority

task_priority Function — Sets the scheduling priority for a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **task_priority** (task_t *task,* int *priority,* boolean_t *change_threads)*;

DESCRIPTION

The **task_priority** function sets the scheduling priority for *task*. The priority of a task is used only when creating new threads. A new thread's priority is set to that of the enclosing task's priority. Changing the priority of a task does not affect the priority of the enclosed threads unless *change_threads* is TRUE. If this priority change violates the maximum priority of some threads, as many threads as possible will be changed and an error code will be returned.

PARAMETERS

task
: The task whose scheduling priority is to be set.

priority
: The new priority for the task.

change_threads
: True if priority of existing threads within the task should also be changed.

RETURN VALUE

KERN_SUCCESS
: The priority has been set.

KERN_INVALID_ARGUMENT
: *task* is not a valid task, or the priority value is out of range for priority values.

KERN_FAILURE
: *change_threads* was TRUE and the attempt to change the priority of some existing thread within the task failed because the new priority would violate that thread's maximum priority.

RELATED INFORMATION

Functions: **thread_max_priority, thread_priority, processor_set_max_priority**.

task_resume

task_resume Function — Resume a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_resume** (task_t *target_task*)

DESCRIPTION

The **task_resume** function decrements the suspend count for *target_task*. If the task's suspend count goes to zero, the function resumes any suspended threads within the task. To resume a given thread, the thread's own suspend count must also be zero.

PARAMETERS

target_task
: The task to be resumed.

RETURN VALUE

KERN_SUCCESS
: The task's suspend count has been decremented.

KERN_FAILURE
: The task's suspend count is already at zero. A suspend count must be either zero or positive.

KERN_INVALID_ARGUMENT
: *target_task* is not a valid task.

RELATED INFORMATION

Functions: **task_create**, **task_info**, **task_suspend**, **task_terminate**, **thread_info**, **thread_resume**, **thread_suspend**.

task_set_bootstrap_port

task_set_bootstrap_port Macro— Sets the bootstrap port for a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_set_bootstrap_port**(task_t *task,* mach_port_t *special_port)*;

DESCRIPTION

The **task_set_bootstrap_port** macro sets the bootstrap port belonging to *task*. It is a short-hand for **task_set_special_port** (*task*, TASK_BOOTSTRAP_PORT, *special_port*).

PARAMETERS

task
: The task for which to set the port.

special_port
: The value for the port.

RETURN VALUE

KERN_SUCCESS
: The port was set.

KERN_INVALID_ARGUMENT
: *task* is not a valid task.

RELATED INFORMATION

Functions: **task_set_special_port**.

task_set_emulation

task_set_emulation Function — Establish a user-level handler for a system call.

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_set_emulation** (task_t *task,* vm_address_t *routine_entry_pt,* int *syscall_number)*;

DESCRIPTION

The **task_set_emulation** function establishes a handler within the task for a particular system call. When a thread executes a system call with this particular number, the system call will be redirected to the specified routine within the task's address space. This is expected to be an address within the transparent emulation library.

These emulation handler addresses are inherited by child processes.

PARAMETERS

task
: The task for which to establish the system call handler.

routine_entry_pt
: The address within the task of the handler for this particular system call.

syscall_number
: The number of the system call to be handled by this handler.

RETURN VALUE

KERN_SUCCESS
: The emulation handler was set.

EML_BAD_TASK
: *task* is not a valid task.

EML_BAD_CNT
: *syscall_number* is not an allowed system call number.

task_set_exception_port

task_set_exception_port Macro— Sets the exception port for a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_set_exception_port**(task_t *task,* mach_port_t *special_port)*;

DESCRIPTION

The **task_set_exception_port** macro sets the exception port belonging to *task*. It is a short-hand for **task_set_special_port** (*task*, TASK_EXCEPTION_PORT, *special_port*).

PARAMETERS

task
The task for which to set the port.

special_port
The value for the port.

RETURN VALUE

KERN_SUCCESS
The port was set.

KERN_INVALID_ARGUMENT
task is not a valid task.

RELATED INFORMATION

Functions: **task_set_special_port**.

task_set_kernel_port

task_set_kernel_port Macro— Sets the kernel port for a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/task_special_ports.h>**)

SYNOPSIS

kern_return_t **task_set_kernel_port**(task_t *task,* mach_port_t *special_port)*;

DESCRIPTION

The **task_set_kernel_port** macro sets the exception port belonging to *task*. It is a short-hand for **task_set_special_port** (*task*, TASK_KERNEL_PORT, *special_port*).

PARAMETERS

task
The task for which to set the port.

special_port
The value for the port.

RETURN VALUE

KERN_SUCCESS
The port was set.

KERN_INVALID_ARGUMENT
task is not a valid task.

RELATED INFORMATION

Functions: **task_set_special_port**.

task_set_special_port

task_set_special_port Function — Sets a special port for a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_set_special_port**(task_t *task,* int *which_port,*
mach_port_t *special_port)*;

DESCRIPTION

The **task_set_special_port** function sets a special port belonging to *task*.

PARAMETERS

task
The task for which to set the port.

which_port
The special port to be set. Valid values are:

TASK_BOOTSTRAP_PORT
The task's bootstrap port. Used to send messages requesting return of other system service ports.

TASK_EXCEPTION_PORT
The task's exception port. Used to receive exception messages from the kernel.

TASK_KERNEL_PORT
The task's kernel port. Used by the kernel to receive messages from the task.

special_port
The value for the port.

RETURN VALUE

KERN_SUCCESS
The port was set.

KERN_INVALID_ARGUMENT
task is not a valid task or *which_port* is not a valid port selector.

RELATED INFORMATION

Functions: **task_create**, **task_get_special_port**, **exception_raise**, **mach_task_self**, **thread_get_special_port**, **thread_set_special_port**.

task_suspend

task_suspend Function — Suspends a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_suspend** (task_t *target_task)*;

DESCRIPTION

The **task_suspend** function increments the suspend count for *target_task* and stops all threads within the task. As long as the suspend count is positive, no newly-created threads can execute. The function does not return until all of the task's threads have been suspended.

To resume a suspended task and its threads, use **task_resume**. If the suspend count is greater than one, you must issue **task_resume** that number of times.

PARAMETERS

target_task
: The task to be suspended.

RETURN VALUE

KERN_SUCCESS
: The task has been suspended.

KERN_INVALID_ARGUMENT
: *target_task* is not a valid task.

RELATED INFORMATION

Functions: **task_create**, **task_info**, **task_resume**, **task_terminate**, **thread_-suspend**.

task_terminate

task_terminate Function — Destroys a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_terminate** (task_t *target_task)*;

DESCRIPTION

The **task_terminate** function kills *target_task* and all its threads, if any. The kernel frees all resources that are in use by the task. The kernel destroys any port for which the task holds the receive right.

PARAMETERS

target_task
The task to be destroyed.

RETURN VALUE

KERN_SUCCESS
The task has been killed.

KERN_INVALID_ARGUMENT
target_task is not a valid task.

RELATED INFORMATION

Functions: **task_create**, **task_suspend**, **task_resume**, **thread_terminate**, **thread_suspend**.

task_threads

task_threads — Returns a list of the threads within a task

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_interface.h>**)

SYNOPSIS

kern_return_t **task_threads** (task_t *target_task,* thread_array_t **thread_list,*
mach_msg_type_number_t **thread_count)*;

DESCRIPTION

The **task_threads** function returns a list of the threads within *target_task*. The calling task or thread also receives a send right to the kernel port for each listed thread.

PARAMETERS

target_task
The task for which the thread list is to be returned.

thread_list
[out array] The returned list of threads within *target_task*, in no particular order.

thread_count
[out] The returned count of threads in *thread_list*.

NOTES

The function returns *thread_list* as an array, automatically allocated as if via **vm_allocate**. It is good practice to de-allocate the array (see **vm_deallocate**) when you finish with the data.

RETURN VALUE

KERN_SUCCESS
The list of threads has been returned.

KERN_INVALID_ARGUMENT
target_task is not a valid task.

RELATED INFORMATION

Functions: **thread_create**, **thread_terminate**, **thread_suspend**.

CHAPTER 8

Host Interface Specifications

This chapter discusses the specifics of the kernel's host interfaces. Included are functions that return status information for a host, such as kernel statistics.

Note that hosts are named both by a name port, which allows the holder to request information about the host, and a control port, which provides full control access. The control port for a host is provided to the bootstrap task for that host.

host_adjust_time

host_adjust_time Function —Gradually change the time

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_adjust_time**(host_priv_t *host_priv,* time_value_t *new_adjustment,* time_value_t **old_adjustment)*;

DESCRIPTION

The **host_adjust_time** function arranges for the time on a specified host to be gradually changed by an adjustment value.

PARAMETERS

host_priv
: The control port the host for which the time is to be set.

new_adjustment
: New adjustment value.

old_adjustment
: [out] Old adjustment value.

NOTES

This function is not yet implemented.

RETURN VALUE

KERN_FAILURE
: This function is not yet implemented.

KERN_NO_ACCESS
: The supplied host port is not the privileged host port.

RELATED INFORMATION

Functions: **host_get_time**, **host_set_time**.

host_get_time

host_get_time Function —Return the current time.

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_get_time** (host_t *host,* time_value_t **current_time)*;

DESCRIPTION

The **host_get_time** function returns the current time as seen by that host.

PARAMETERS

host
The name port the host for which the time is to be set.

current_time
[out] Returned time value.

RETURN VALUE

KERN_SUCCESS
The current time is returned.

RELATED INFORMATION

Functions: **host_adjust_time**, **host_set_time**.

host_info

host_info Function — Returns information about a host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_info** (host_t *host,* int *flavor,* host_info_t *host_info,*
mach_msg_type_number_t **host_infoCnt)*;

DESCRIPTION

The **host_info** function returns selected information about a host, as specified by *flavor*. *host_info* is an array of integers that is supplied by the caller, and filled with the specified information. *host_infoCnt* is supplied as the maximum number of integers in *host_info*. On return, it contains the actual number of integers in *host_info*.

Basic information is defined by HOST_BASIC_INFO. Processor slots of the active (available) processors are defined by HOST_PROCESSOR_SLOTS. Additional information of interest to schedulers is defined by HOST_LOAD_INFO and HOST_SCHED_INFO.

PARAMETERS

host
: The name port for the host for which information is to be obtained.

flavor
: The type of statistics desired. Currently, HOST_BASIC_INFO, HOST_LOAD_INFO, HOST_PROCESSOR_SLOTS and HOST_SCHED_INFO are defined.

host_info
: [in/out] Statistics about the specified host. The relevant structures are **host_basic_info, host_load_info** and **host_sched_info**. In the case of HOST_PROCESSOR_SLOTS, the return value is an array of processor slot numbers for active processors.

host_infoCnt
: [in/out] Size of the information structure, in units of sizeof(int). This should be HOST_BASIC_INFO_COUNT (for HOST_BASIC_INFO), HOST_SCHED_INFO_COUNT (for HOST_SCHED_INFO), HOST_LOAD_INFO_COUNT (for HOST_LOAD_INFO) and the maximum number of CPUs reported by HOST_BASIC_INFO (for HOST_PROCESSOR_SLOTS).

RETURN VALUE

KERN_SUCCESS
The information has been returned.

KERN_INVALID_ARGUMENT
host is not a host port or *flavor* is not recognized.

MIG_ARRAY_TOO_LARGE
Returned info array is too large for *host_info*. *host_info* is filled as much as possible. *host_infoCnt* is set to the number of elements that would be returned if there were enough room.

RELATED INFORMATION

Functions: **host_kernel_version, host_processors, processor_info**.

Data Structures: **host_basic_info, host_load_info, host_sched_info**

host_kernel_version

host_kernel_version Function — Returns kernel version information for a host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_kernel_version** (host_t *host,* kernel_version_t **version)*;

DESCRIPTION

The **host_kernel_version** function returns the version string compiled into the kernel executing on *host* at the time it was built. This describes the version of the kernel. The constant KERNEL_VERSION_MAX should be used to dimension storage for the returned string if the **kernel_version_t** declaration is not used.

PARAMETERS

host
: The name port for the host for which information is to be obtained.

version
: [out] Character string describing the kernel version executing on *host*

RETURN VALUE

KERN_SUCCESS
: The information has been returned.

KERN_INVALID_ARGUMENT
: *host* is not a host port.

KERN_INVALID_ADDRESS
: *version* points to inaccessible memory.

RELATED INFORMATION

Functions: **host_info, host_ports, host_processors, processor_info**.

host_reboot

host_reboot Function — Reboot this host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_reboot** (host_priv_t *host_priv,* int *options)*;

DESCRIPTION

The **host_reboot** function reboots the specified host.

PARAMETERS

host_priv
: The control port the host to be re-booted.

options
: Reboot options. See **<sys/reboot.h>** for details.

NOTES

If successful, this call will not return.

RETURN VALUE

KERN_NO_ACCESS
: The supplied host port is not the privileged host port.

host_set_time

host_set_time Function — Sets the time

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_set_time** (host_priv_t *host_priv,* time_value_t *new_time)*;

DESCRIPTION

The **host_set_time** function establishes the time on the specified host.

PARAMETERS

host_priv
: The control port for the host for which the time is to be set.

new_time
: Time to be set.

RETURN VALUE

KERN_SUCCESS
: The time is set.

KERN_NO_ACCESS
: The supplied host port is not the privileged host port.

RELATED INFORMATION

Functions: **host_adjust_time**, **host_get_time**.

mach_host_self

mach_host_self Function — Returns the host self port

LIBRARY

libmach.a

#include <mach/mach_traps.h>

SYNOPSIS

mach_port_t **mach_host_self**();

DESCRIPTION

The **mach_host_self** function returns send rights to the current host's name port.

PARAMETERS

None

NOTES

This function is actually a system call.

RETURN VALUE

Send rights to the host's name port.

RELATED INFORMATION

Functions: **host_info**.

CHAPTER 9 Processor Interface Specifications

This chapter discusses the specifics of the kernel's processor and processor set interfaces. This includes functions to control processors, change their assignments, assign tasks and threads to processors, and processor status returning functions.

Note that processor sets have two ports that name them: a name port which allows information to be requested about them, and a control port which allows full access. The control port for a processor set is provided to the creator of the set.

Processors have only a single port that names them. The host control port is needed to obtain these processor ports.

host_processor_set_priv

host_processor_set_priv Function — Returns a processor set control port for a host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_processor_set_priv**(host_priv_t *host_priv,*
processor_set_name_t *set_name,* processor_set_t
**processor_set)*;

DESCRIPTION

The **host_processor_set_priv** function returns send rights for the control port for a specified processor set currently existing on *host_priv.*

PARAMETERS

host_priv
The control port for the host for which the processor set is desired.

set_name
The name port for the processor set desired.

processor_set
[out] The returned processor set control port.

RETURN VALUE

KERN_SUCCESS
The port has been returned.

KERN_INVALID_ARGUMENT
host_priv is not a valid host control port.

RELATED INFORMATION

Functions: **host_processor_sets, processor_set_create, processor_set_-tasks, processor_set_threads**.

host_processor_sets

host_processor_sets Function — Returns processor set ports for a host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_processor_sets** (host_t *host,* processor_set_array_t **processor_set_list,* mach_msg_type_number_t **processor_set_count)*;

DESCRIPTION

The **host_processor_sets** function returns send rights for the name ports for each processor set currently existing on *host*.

PARAMETERS

host
: The name port for the host for which the processor sets are desired.

processor_set_list
: [out array] The set of processor set name ports for those currently existing on *host*; no particular order is guaranteed.

processor_set_count
: [out] The number of processor sets returned.

NOTES

If control ports to the processor sets are needed, use **host_processor_set_priv**.

processor_set_list is automatically allocated by the kernel, as if by **vm_allocate**. It is good practice to **vm_deallocate** this space when it is no longer needed.

RETURN VALUE

KERN_SUCCESS
: The ports have been returned.

KERN_INVALID_ARGUMENT
: *host* is not a valid host.

RELATED INFORMATION

Functions: **host_processor_set_priv, processor_set_create, processor_set_-tasks, processor_set_threads**.

host_processors

host_processors Function — Gets processor ports for a host

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **host_processors**(host_priv_t *host_priv,* processor_array_t **processor_list,* mach_msg_type_number_t **processor_count)*;

DESCRIPTION

The **host_processors** function returns an array of send right ports for each processor existing on *host_priv*.

PARAMETERS

host_priv
: The control port for the desired host.

processor_list
: [out array] The set of processors existing on *host_priv*; no particular order is guaranteed.

processor_count
: [out] The number of ports returned in *processor_list*.

NOTES

processor_list is automatically allocated by the kernel, as if by **vm_allocate**. It is good practice to **vm_deallocate** this space when it is no longer needed.

RETURN VALUE

KERN_SUCCESS
: The list of ports is returned.

KERN_INVALID_ARGUMENT
: *host_priv* is not a privileged host port.

KERN_INVALID_ADDRESS
: *processor_count* points to invalid memory.

RELATED INFORMATION

Functions: **processor_start, processor_exit, processor_info, processor_control**.

processor_assign

processor_assign Function — Assign a processor to a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_assign**(processor_t *processor,* processor_set_t *new_set,* boolean_t *wait)*;

DESCRIPTION

The **processor_assign** function assigns *processor* to the set *new_set*. After the assignment is completed, the processor only executes threads that are assigned to that processor set. Any previous assignment of the processor is nullified. The master processor cannot be reassigned.

The *wait* argument indicates whether the caller should wait for the assignment to be completed or should return immediately. Dedicated kernel threads are used to perform processor assignment, so setting *wait* to FALSE allows assignment requests to be queued and performed quicker, especially if the kernel has more than one dedicated internal thread for processor assignment.

All processors take clock interrupts at all times. Redirection of other device interrupts away from processors assigned to other than the default processor set is machine dependent.

PARAMETERS

processor
: The processor to be assigned.

new_set
: The control port for the processor set into which the processor is to be assigned.

wait
: True if the call should wait for the completion of the assignment.

CAUTIONS

Intermediaries that interpose on ports must be sure to interpose on both ports involved in the call if they interpose on either.

RETURN VALUE

KERN_SUCCESS
: The assignment was performed.

KERN_INVALID_ARGUMENT
: *processor* is not a processor port, or *new_set* is not a processor set port for the same host as *processor*.

RELATED INFORMATION

Functions**: processor_set_create, processor_set_info, task_assign**, **thread_assign**.

processor_control

processor_control Function — Do something to a processor

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_control**(processor_t *processor,* processor_info_t *cmd,* mach_msg_type_number_t *count)*;

DESCRIPTION

The **processor_control** function allows privileged software to control a processor in a multi-processor that so allows it. The interpretation of *cmd* is machine dependent.

PARAMETERS

processor
The processor to be controlled.

cmd
An array containing the command to be applied to the processor.

count
The size of the *cmd* array.

NOTES

These operations are machine dependent. They may do nothing.

RETURN VALUE

KERN_SUCCESS
The operation was performed.

KERN_FAILURE
The operation was not performed. A likely reason is that it is not supported on this processor.

KERN_INVALID_ARGUMENT
processor is not a processor port.

KERN_INVALID_ADDRESS
cmd points to inaccessible memory.

RELATED INFORMATION

Functions: **processor_start, processor_exit, processor_info, host_processors**.

processor_exit

processor_exit Function — Exit a processor

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_exit**(processor_t *processor)*;

DESCRIPTION

The **processor_exit** function allows privileged software to exit a processor in a multi-processor that so allows it. An exited processor is removed from the processor set to which it was assigned and ceases to be active. The interpretation of this operation is machine dependent.

PARAMETERS

processor
The processor to be controlled.

NOTES

This operation is machine dependent. It may do nothing.

CAUTIONS

The ability to restart an exited processor is machine dependent.

RETURN VALUE

KERN_SUCCESS
The operation was performed.

KERN_FAILURE
The operation was not performed. A likely reason is that it is not supported on this processor.

KERN_INVALID_ARGUMENT
processor is not a processor port.

RELATED INFORMATION

Functions**: processor_control, processor_start, processor_info, host_processors**.

processor_get_assignment

processor_get_assignment Function — Get current assignment for a processor

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_get_assignment**(processor_t *processor,*
processor_set_name_t **assigned_set)*;

DESCRIPTION

The **processor_get_assignment** function returns the name port for the processor set to which a desired processor is currently assigned.

PARAMETERS

processor
The processor whose assignment is desired.

new_set
[out] The name port for the processor set to which *processor* is currently assigned.

RETURN VALUE

KERN_SUCCESS
The processor set name was returned.

KERN_INVALID_ARGUMENT
processor is not a processor port.

KERN_INVALID_ADDRESS
assigned_set points to inaccessible memory.

KERN_FAILURE
processor is either shut down of off-line.

RELATED INFORMATION

Functions**: processor_assign, processor_set_create, processor_info, task_assign**, **thread_assign**.

processor_info

processor_info Function — Returns information about a processor.

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_info** (processor_t *processor,* int *flavor,* host_t **host,*
processor_info_t *processor_info,*
mach_msg_type_number_t **processor_infoCnt)*;

DESCRIPTION

The **processor_info** function returns selected information for a processor as an array, as specified by *flavor*. *processor_info* is an array of integers that is supplied by the caller, and filled with the specified information. *processor_infoCnt* is supplied as the maximum number of integers in *processor_info*. On return, it contains the actual number of integers in *processor_info*.

Basic information is defined by PROCESSOR_BASIC_INFO. Additional information is defined by machine-dependent values of *flavor.*

PARAMETERS

processor
A processor port for which information is desired.

flavor
The type of information requested. Currently, only PROCESSOR_-BASIC_INFO is defined.

host
[out] The host on which the processor resides. This is the non-privileged host port.

processor_info
[in/out] Information about the processor.

processor_infoCnt
[in/out] Size of the info structure, in units of sizeof(int). This should be PROCESSOR_BASIC_INFO_COUNT (for PROCESSOR_BASIC_INFO).

RETURN VALUE

KERN_SUCCESS
The information has been returned.

KERN_INVALID_ARGUMENT
processor is not a processor port, or *flavor* is not recognized.

MIG_ARRAY_TOO_LARGE
Returned info array is too large for *processor_info*. *processor_info* is filled as much as possible. *processor_infoCnt* is set to the number of elements that would be returned if there were enough room.

RELATED INFORMATION

Functions**: processor_start, processor_exit, processor_control, host_processors**.

Data Structures: **processor_basic_info.**

processor_set_create

processor_set_create Function — Creates a new processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_create**(host_t *host,* processor_set_t **new_set,*
processor_set_name_t **new_name)*;

DESCRIPTION

The **processor_set_create** function creates a new processor set and returns the two ports associated with it. The port returned in *new_set* is the control port representing the set. It is used to perform operations such as assigning processors, tasks or threads. The port returned in *new_name* is the name port which identifies the set, and is used to obtain information about the set.

PARAMETERS

host
: The name port for the host on which the set is to be created.

new_set
: [out] Control port used for performing operations on the new set.

new_name
: [out] Name port used to identify the new set and obtain information about it.

RETURN VALUE

KERN_SUCCESS
: The set was created.

KERN_INVALID_ARGUMENT
: *host* is not a host port.

KERN_INVALID_ADDRESS
: *new_set* and/or *new_name* point to inaccessible memory.

RELATED INFORMATION

Functions**: processor_set_destroy, processor_set_info, processor_assign, task_assign**, **thread_assign**.

processor_set_default

processor_set_default Function — Returns the default processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_default**(host_t *host,* processor_set_name_t **default_set)*;

DESCRIPTION

The **processor_set_default** function returns the name port for the default processor set for the specified host. The default processor set is used by all threads, tasks and processors that are not explicitly assigned to other sets. The port returned can be used to obtain information about this set (such as how many threads are assigned to it). It cannot be used to perform operations on the set.

PARAMETERS

host
: The name port for the host for which the default processor set is desired.

default_set
: [out] The returned name port for the default processor set.

RETURN VALUE

KERN_SUCCESS
: The default set has been returned.

KERN_INVALID_ARGUMENT
: *host* was not a host.

KERN_INVALID_ADDRESS
: *default_set* points to inaccessible memory.

RELATED INFORMATION

Functions**: processor_set_info, thread_assign, task_assign**.

processor_set_destroy

processor_set_destroy Function — Destroys a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_destroy**(processor_set_t *processor_set)*;

DESCRIPTION

The **processor_set_destroy** function destroys the specified processor set. Any assigned processors, tasks or threads are re-assigned to the default set. The object port (not the name port) for the processor set is required.

PARAMETERS

processor_set
The control port for the processor set to be destroyed.

RETURN VALUE

KERN_SUCCESS
The set was destroyed.

KERN_FAILURE
An attempt was made to destroy the default processor set.

KERN_INVALID_ARGUMENT
processor_set is not a processor set control port.

RELATED INFORMATION

Functions**: processor_set_create, processor_assign, task_assign**, **thread_assign**.

processor_set_info

processor_set_info Function — Returns information about a processor set.

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_info**(processor_set_t *processor_set,* int *flavor,*
host_t **host,* processor_set_info_t
processor_set_info, mach_msg_type_number_t
**infoCnt)*;

DESCRIPTION

The **processor_set_info** function returns selected information for a processor set as an array, as specified by *flavor*. *processor_set_info* is an array of integers that is supplied by the caller, and filled with the specified information. *infoCnt* is supplied as the maximum number of integers in *processor_set_info*. On return, it contains the actual number of integers in *processor_set_info*.

Basic information is defined by PROCESSOR_SET_BASIC_INFO. Scheduling information is given by PROCESSOR_SET_SCHED_INFO.

PARAMETERS

processor_set
: A processor set name or control port for which information is desired.

flavor
: The type of information requested. Currently, PROCESSOR_SET_-BASIC_INFO and PROCESSOR_SET_SCHED_INFO are defined.

host
: [out] The name port for the host on which the processor resides.

processor_set_info
: [in/out] Information about the processor set.

infoCnt
: [in/out] Size of the info structure, in units of sizeof(int). This should be PROCESSOR_SET_BASIC_INFO_COUNT (for PROCESSOR_SET_BASIC_INFO) and PROCESSOR_SET_SCHED_INFO_COUNT (for PROCESSOR_SCHED_INFO).

RETURN VALUE

KERN_SUCCESS
: The information has been returned.

KERN_INVALID_ARGUMENT
: *processor_set* is not a processor set port, or *flavor* is not recognized.

MIG_ARRAY_TOO_LARGE
: Returned info array is too large for *processor_set_info*. *processor_set_info* is filled as much as possible. *infoCnt* is set to the number of elements that would be returned if there were enough room.

RELATED INFORMATION

Functions**: processor_set_create, processor_set_default, processor_assign, task_assign**, **thread_assign**.

Data Structures: **processor_set_basic_info**, **processor_set_sched_info.**

processor_set_max_priority

processor_set_max_priority Function — Sets the maximum scheduling priority for a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_max_priority**(processor_set_t *processor_set,*
int *priority,* boolean_t *change_threads)*;

DESCRIPTION

The **processor_set_max_priority** function sets the maximum scheduling priority for *processor_set*. The maximum priority of a processor set is used only when creating new threads. A new thread's maximum priority is set to that of its assigned processor set. When assigned to a processor set, a thread's maximum priority is reduced, if necessary, to that of its new processor set; its current priority is also reduced, as needed. Changing the maximum priority of a processor set does not affect the priority of the currently assigned threads unless *change_threads* is TRUE. If this priority change violates the maximum priority of some threads, their maximum priorities will be reduced to match.

PARAMETERS

processor_set
: The control port for the processor set whose maximum scheduling priority is to be set.

priority
: The new priority for the processor set.

change_threads
: True if the maximum priority of existing threads assigned to this processor set should also be changed.

RETURN VALUE

KERN_SUCCESS
: The priority has been set.

KERN_INVALID_ARGUMENT
: *processor_set* is not a valid processor set, or the priority value is out of range for priority values.

RELATED INFORMATION

Functions: **thread_max_priority, thread_priority, thread_assign**.

processor_set_policy_disable

processor_set_policy_disable Function — Disables a scheduling policy for a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_policy_disable**(processor_set_t *processor_set,*
int *policy,* boolean_t *change_threads)*;

DESCRIPTION

The **processor_set_policy_disable** function restricts the set of scheduling policies allowed for *processor_set*. The set of scheduling policies allowed for a processor set is the set of policies allowed to be set for threads assigned to that processor set. The current set of permitted policies can be obtained from **processor_set_info**. Timesharing may not be forbidden for any processor set. This is a compromise to reduce the complexity of the assign operation; any thread whose policy is forbidden by its target processor set has its policy reset to timesharing. Disabling a scheduling policy for a processor set has no effect on threads currently assigned to that processor set unless *change_threads* is TRUE, in which case their policies will be reset to timesharing.

PARAMETERS

processor_set
: The control port for the processor set for which a scheduling policy is to be disabled.

policy
: Policy to be disabled. The values currently defined are POLICY_TIMESHARE and POLICY_FIXEDPRI.

change_threads
: If true, causes the scheduling policy for all threads currently running with *policy* to POLICY_TIMESHARE.

RETURN VALUE

KERN_SUCCESS
: The policy has been disabled.

KERN_INVALID_ARGUMENT
: *processor_set* is not a valid processor set, or *policy* is not a recognized scheduling policy value, or an attempt was made to disable timesharing.

RELATED INFORMATION

Functions: **processor_set_policy_enable, thread_policy**.

processor_set_policy_enable

processor_set_policy_enable Function — Enables a scheduling policy for a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_policy_enable**(processor_set_t *processor_set,*
int *policy)*;

DESCRIPTION

The **processor_set_policy_enable** function extends the set of scheduling policies allowed for *processor_set*. The set of scheduling policies allowed for a processor set is the set of policies allowed to be set for threads assigned to that processor set. The current set of permitted policies can be obtained from **processor_set_info**.

PARAMETERS

processor_set
: The control port for the processor set for which a scheduling policy is to be enabled.

policy
: Policy to be enabled. The values currently defined are POLICY_-TIMESHARE and POLICY_FIXEDPRI.

RETURN VALUE

KERN_SUCCESS
: The policy has been enabled.

KERN_INVALID_ARGUMENT
: *processor_set* is not a valid processor set, or *policy* is not a recognized scheduling policy value.

RELATED INFORMATION

Functions: **processor_set_policy_disable, thread_policy**.

processor_set_tasks

processor_set_tasks Function — Returns a list of tasks assigned to a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_tasks**(processor_set_t *processor_set,*
task_array_t **task_list,* mach_msg_type_number_t
**task_count)*;

DESCRIPTION

The **processor_set_tasks** function returns send rights to the kernel ports for each task currently assigned to *processor_set*.

PARAMETERS

processor_set
A processor set control port for which information is desired.

task_list
[out array] The returned set of ports naming the tasks currently assigned to *processor_set*.

task_count
[out] The number of tasks returned in *task_list*.

NOTES

task_list is automatically allocated by the kernel, as if by **vm_allocate**. It is good practice to **vm_deallocate** this space when it is no longer needed.

RETURN VALUE

KERN_SUCCESS
The information has been returned.

KERN_INVALID_ARGUMENT
processor_set is not a processor set port.

RELATED INFORMATION

Functions**: processor_set_threads, task_assign**, **thread_assign**.

processor_set_threads

processor_set_threads Function — Returns a list of threads assigned to a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_set_threads**(processor_set_t *processor_set,*
thread_array_t **thread_list,*
mach_msg_type_number_t **thread_count)*;

DESCRIPTION

The **processor_set_threads** function returns send rights to the kernel ports for each thread currently assigned to *processor_set*.

PARAMETERS

processor_set
A processor set control port for which information is desired.

thread_list
[out array] The returned set of ports naming the threads currently assigned to *processor_set*.

thread_count
[out] The number of threads returned in *thread_list*.

NOTES

thread_list is automatically allocated by the kernel, as if by **vm_allocate**. It is good practice to **vm_deallocate** this space when it is no longer needed.

RETURN VALUE

KERN_SUCCESS
The information has been returned.

KERN_INVALID_ARGUMENT
processor_set is not a processor set port.

RELATED INFORMATION

Functions**: processor_set_tasks, task_assign**, **thread_assign**.

processor_start

processor_start Function — Start a processor

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **processor_start**(processor_t *processor)*;

DESCRIPTION

The **processor_start** function allows privileged software to start a processor in a multi-processor that so allows it. A newly started processor is assigned to the default processor set. The interpretation of this operation is machine dependent.

PARAMETERS

processor
The processor to be controlled.

NOTES

This operation is machine dependent. It may do nothing.

CAUTIONS

The ability to restart an exited processor is machine dependent.

RETURN VALUE

KERN_SUCCESS
The operation was performed.

KERN_FAILURE
The operation was not performed. A likely reason is that it is not supported on this processor.

KERN_INVALID_ARGUMENT
processor is not a processor port.

RELATED INFORMATION

Functions**: processor_control, processor_exit, processor_info, host_processors**.

task_assign

task_assign Function — Assign a task to a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **task_assign** (task_t *task,* processor_set_t *processor_set,*
boolean_t *assign_threads)*;

DESCRIPTION

The **task_assign** function assigns *task* to the set *processor_set*. After the assignment is completed, newly created threads within this task will be assigned to this processor set. Any previous assignment of the task is nullified.

If *assign_threads* is TRUE, existing threads within the task will also be assigned to the processor set.

PARAMETERS

task
: The task to be assigned.

processor_set
: The control port for the processor set into which the task is to be assigned.

assign_threads
: True if this assignment should apply as well to the threads within the task.

RETURN VALUE

KERN_SUCCESS
: The assignment was performed.

KERN_INVALID_ARGUMENT
: *task* is not a task port, or *processor_set* is not a processor set port for the same host as *task.*

RELATED INFORMATION

Functions**: task_assign_default, task_get_assignment, processor_set_create, processor_set_info**, **processor_assign**, **thread_assign**.

task_assign_default

task_assign_default Function — Assign a task to the default processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **task_assign_default** (task_t *task,* boolean_t *assign_threads)*;

DESCRIPTION

The **task_assign_default** function assigns *task* to the default processor set. After the assignment is completed, newly created threads within this task will be assigned to this processor set. Any previous assignment of the task is nullified.

If *assign_threads* is TRUE, existing threads within the task will also be assigned to the processor set.

This variant of **task_assign** exists because the control port for the default processor set is privileged, and therefore not available to most tasks.

PARAMETERS

task
: The task to be assigned.

assign_threads
: True if this assignment should apply as well to the threads within the task.

RETURN VALUE

KERN_SUCCESS
: The assignment was performed.

KERN_INVALID_ARGUMENT
: *task* is not a task port.

RELATED INFORMATION

Functions**: task_assign, task_get_assignment, processor_set_create, processor_set_info, thread_assign**, **processor_assign**.

task_get_assignment

task_get_assignment Function — Returns the processor set to which a task is assigned

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **task_get_assignment** (task_t *task,* processor_set_name_t **processor_set)*;

DESCRIPTION

The **task_get_assignment** function returns the name port to the processor set to which *task* is currently assigned. This port can only be used to obtain information about the processor set.

PARAMETERS

task
: The task whose assignment is desired.

processor_set
: [out] The name port for the processor set into which the task is assigned.

RETURN VALUE

KERN_SUCCESS
: The assigned set was returned.

KERN_INVALID_ARGUMENT
: *task* is not a task port.

KERN_INVALID_ADDRESS
: *processor_set* points to inaccessible memory.

RELATED INFORMATION

Functions**: task_assign, task_assign_default, processor_set_create, processor_set_info, thread_assign**, **processor_assign**.

thread_assign

thread_assign Function — Assign a thread to a processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_assign** (thread_t *thread,* processor_set_t *processor_set)*;

DESCRIPTION

The **thread_assign** function assigns *thread* to the set *processor_set*. After the assignment is completed, the thread executes only on processors that are assigned to that processor set. Any previous assignment of the thread is nullified.

PARAMETERS

thread
: The thread to be assigned.

processor_set
: The name port for the processor set into which the thread is to be assigned.

RETURN VALUE

KERN_SUCCESS
: The assignment was performed.

KERN_INVALID_ARGUMENT
: *thread* is not a thread port, or *processor_set* is not a processor set port for the same host as *thread.*

RELATED INFORMATION

Functions**: thread_assign_default, thread_get_assignment, processor_set_create, processor_set_info, task_assign**, **processor_assign**.

thread_assign_default

thread_assign_default Function — Assign a thread to the default processor set

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_assign_default**(thread_t *thread)*;

DESCRIPTION

The **thread_assign_default** function assigns *thread* to the default processor set. After the assignment is completed, the thread executes only on processors that are assigned to that processor set. Any previous assignment of the thread is nullified. This variant of **thread_assign** exists because the control port for the default processor set is privileged, and therefore not available to most tasks.

PARAMETERS

thread
: The thread to be assigned.

RETURN VALUE

KERN_SUCCESS
: The assignment was performed.

KERN_INVALID_ARGUMENT
: *thread* is not a thread port.

RELATED INFORMATION

Functions**: thread_assign, thread_get_assignment, processor_set_create, processor_set_info, task_assign**, **processor_assign**.

thread_get_assignment

thread_get_assignment Function — Returns the processor set to which a thread is assigned

LIBRARY

libmach.a

#include <mach.h> (**<mach/mach_host.h>**)

SYNOPSIS

kern_return_t **thread_get_assignment**(thread_t *thread,*
processor_set_name_t **processor_set)*;

DESCRIPTION

The **thread_get_assignment** function returns the name port to the processor set to which *thread* is currently assigned. This port can only be used to obtain information about the processor set.

PARAMETERS

thread
: The thread whose assignment is desired.

processor_set
: [out] The name port for the processor set into which the thread is assigned.

RETURN VALUE

KERN_SUCCESS
: The assigned set was returned.

KERN_INVALID_ARGUMENT
: *thread* is not a thread port.

KERN_INVALID_ADDRESS
: *processor_set* points to inaccessible memory.

RELATED INFORMATION

Functions**: thread_assign, thread_assign_default, processor_set_create, processor_set_info, task_assign**, **processor_assign**.

CHAPTER 10

Device Interface Specifications

This chapter discusses the specifics of the kernel’s device interfaces. These interfaces provide read, write and status interfaces to devices.

device_close

device_close Function — De-establish a connection to a device.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_close** (device_t *device)*;

DESCRIPTION

The **device_close** function decrements the open count for the named device. If this count reaches zero, the close operation of the device driver is invoked, closing the device.

PARAMETERS

device
: A device port to the device to be closed.

RETURN VALUE

D_SUCCESS
: Device was closed.

D_NO_SUCH_DEVICE
: *device* does not name a device port.

RELATED INFORMATION

Functions: **device_open**, **device_open_reply**, **device_open_request**.

device_get_status

device_get_status Function — Return the current device status

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_get_status** (device_t *device,* int *flavor,* dev_status_t *status,* mach_msg_type_number_t **status_count)*;

DESCRIPTION

The **device_get_status** function returns status information pertaining to an open device. The possible values for *flavor* as well as the meaning of the returned status information is device dependent.

PARAMETERS

device
: A device port to the device to be interrogated.

flavor
: The type of status information requested.

status
: [out] The returned device status.

status_count
: [out] The size of the returned device status.

RETURN VALUE

D_SUCCESS
: Status was returned.

D_NO_SUCH_DEVICE
: Device is not open or operational.

RELATED INFORMATION

Functions: **device_set_status**.

device_map

device_map Function — Establish a memory manager representing a device

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_map** (device_t *device,* vm_prot_t *prot,* vm_offset_t *offset,* vm_size_t *size,* memory_object_t **pager,* int *unmap)*;

DESCRIPTION

The **device_map** function establishes a memory manager that presents a memory object representing a device. The resulting port is suitable for use as the pager port in a **vm_map** call. This call is device dependent.

PARAMETERS

device
: A device port to the device to be mapped.

prot
: Protection for the device memory.

offset
: An offset within the device memory object, in bytes.

size
: The size of the device memory object.

pager
: [out] The returned abstract memory object port to a memory manager that represents the device.

unmap
: Currently unused.

NOTES

Port rights are maintained as follows:

Abstract memory object port:
: The device pager has all rights.

Memory cache control port:
: The device pager has only send rights.

Memory cache name port:
: The device pager has only send rights. The name port is not even recorded.

Regardless how the object is created, the control and name ports are created by the kernel and passed through the memory management interface.

CAUTIONS

The device pager assumes that access to its memory objects will not be propagated to more that one host, and therefore provides no consistency guarantees beyond those made by the kernel.

In the event that more than one host attempts to use a device memory object, the device pager will only record the last set of port names. [This can happen with only one host if a new mapping is being established while termination of all previous mappings is taking place.] Currently, the device pager assumes that its clients adhere to the initialization and termination protocols in the memory management interface; otherwise, port rights or out-of-line memory from erroneous messages may be allowed to accumulate.

RETURN VALUE

KERN_SUCCESS
: The device map is established.

D_NO_SUCH_DEVICE
: The device is not open or not operational.

RELATED INFORMATION

Functions: **vm_map**.

device_open

device_open Function — Establish a connection to a device.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_open** (mach_port_t *master_port,* dev_mode_t *mode,*
dev_name_t *name,* device_t **device)*;

DESCRIPTION

The **device_open** function opens a device object. The open operation of the device is invoked, if the device is not already open. The open count for the device is incremented.

PARAMETERS

master_port
: The master device port. This port is provided to the bootstrap task.

mode
: Opening mode. This is the bit-wise OR of the following values:

D_READ
: Read access

D_WRITE
: Write access

D_NODELAY
: Do not delay on open

name
: Name of the device to open.

device
: [out] The returned device port.

RETURN VALUE

D_SUCCESS
: Device was opened.

D_INVALID_OPERATION
: *master_port* is not the master device port.

D_WOULD_BLOCK
The device is busy, but D_NOWAIT was specified in *mode*.

D_ALREADY_OPEN
The device is already open in a mode incompatible with *mode*.

D_NO_SUCH_DEVICE
name does not name a known device.

D_DEVICE_DOWN
The device has been shut down.

RELATED INFORMATION

Functions: **device_close**, **device_open_reply**, **device_open_request**.

device_open_reply

device_open_reply Server Interface — Receive the result from a device open call.

SYNOPSIS

kern_return_t **device_open_reply**(mach_port_t *reply_port,* kern_return_t *return_code,* mach_port_t *device_port)*;

DESCRIPTION

A **device_open_reply** function is called as the result of a message from the kernel corresponding to the completion of an (asynchronous) device open call.

PARAMETERS

reply_port
: The requestor of the open operation.

return_code
: Status of the open.

device_port
: [out] The returned device port.

RETURN VALUE

KERN_SUCCESS
: Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_close**, **device_open**, **device_open_request**.

device_open_request

device_open_request Function — Asynchronously establishes a connection to a device.

LIBRARY

libmach.a

#include <device_request.h>

SYNOPSIS

kern_return_t **device_open_request**(mach_port_t *device_server_port,*
mach_port_t *reply_port,* dev_mode_t *mode,*
dev_name_t *name)*;

DESCRIPTION

The **device_open_request** function opens a device object. Unlike **device_-open**, this call returns immediately, without waiting for an I/O necessary to open the device. The result of the open operation is received as a message to the supplied *reply_port*. See **device_open_reply**.

PARAMETERS

device_server_port
The master device port. This port is provided to the bootstrap task.

reply_port
The port to which a reply is to be sent when the device is open.

mode
Opening mode. This is the bit-wise OR of the following values:

D_READ
Read access

D_WRITE
Write access

D_NODELAY
Do not delay on open

name
Name of the device to open.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_close**, **device_open**, **device_open_reply**.

device_read

device_read Function — Read a sequence of bytes from a device object.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_read** (device_t *device,* dev_mode_t *mode,* recnum_t *recnum,* int *bytes_wanted,* io_buf_ptr_t **data,* mach_msg_type_number_t **data_count)*;

DESCRIPTION

The **device_read** function reads a sequence of bytes from a device object. The meaning of *recnum* as well as the specific operation performed is device dependent.

PARAMETERS

device
A device port to the device to be read.

mode
I/O mode value. Meaningful options are:

D_NOWAIT
Do not wait if data is unavailable.

recnum
Record number to be read.

bytes_wanted
Size of data transfer.

data
[out array] Returned data bytes.

data_count
[out] Number of returned data bytes.

RETURN VALUE

D_SUCCESS
Data was read.

D_NO_SUCH_DEVICE
The device is dead or not completely open.

RELATED INFORMATION

Functions: **device_read_inband**, **device_read_inband_reply**, **device_read_-inband_request**, **device_read_reply**, **device_read_request**.

device_read_inband

device_read_inband Function — Read a sequence of bytes “inband” from a device object.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_read_inband**(device_t *device,* dev_mode_t *mode,*
recnum_t *recnum,* int *bytes_wanted,*
io_buf_ptr_inband_t **data,*
mach_msg_type_number_t **data_count)*;

DESCRIPTION

The **device_read** function reads a sequence of bytes from a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call differs from **device_read** in that the returned bytes are returned “inband” in the reply IPC message.

PARAMETERS

device
A device port to the device to be read.

mode
I/O mode value. Meaningful options are:

D_NOWAIT
Do not wait if data is unavailable.

recnum
Record number to be read.

bytes_wanted
Size of data transfer.

data
[out array] Returned data bytes.

data_count
[out] Number of returned data bytes.

RETURN VALUE

D_SUCCESS
Data was read.

D_NO_SUCH_DEVICE
: The device is dead or not completely open.

RELATED INFORMATION

Functions: **device_read**, **device_read_inband_reply**, **device_read_inband_request**, **device_read_reply**, **device_read_request**.

device_read_reply

device_read_reply Server Interface — Receives the reply message from an asynchronous read request.

SYNOPSIS

kern_return_t **device_read_reply**(mach_port_t *reply_port,* kern_return_t *return_code,* io_buf_ptr_t *data,* mach_msg_type_number_t *data_count)*;

DESCRIPTION

A **device_read_reply** function is called as the result of a message from the kernel returning the data requested in a **device_read_request** call.

PARAMETERS

reply_port
The port to which the reply message was sent.

return_code
The return status code from the read.

data
Data bytes to return.

data_count
Number of data bytes to return.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_read_inband**, **device_read_inband_reply**, **device_read_-inband_request**, **device_read**, **device_read_request**.

device_read_reply_inband

device_read_reply_inband Server Interface — Receives the reply message from an asynchronous "inband" read request.

SYNOPSIS

kern_return_t **device_read_reply_inband**(mach_port_t *reply_port,*
kern_return_t *return_code,* io_buf_ptr_inband_t
data, mach_msg_type_number_t *data_count)*;

DESCRIPTION

A **device_read_reply_inband** function is called as the result of a kernel message returning the data requested in a **device_read_request_inband** call. It differs from **device_read_reply** in that the data bytes are returned "inband" in the IPC message.

PARAMETERS

reply_port
The port to which the reply message was sent.

return_code
The return status code from the read.

data
Data bytes to return.

data_count
Number of data bytes to return.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_read_inband**,**device_read_inband_request**, **device_-read**, **device_read_reply, device_read_request**.

device_read_request

device_read_request Function — Asynchronously read a sequence of bytes from a device object.

LIBRARY

libmach.a

#include <device_request.h>

SYNOPSIS

kern_return_t **device_read_request**(device_t *device,* mach_port_t *reply_port,* dev_mode_t *mode,* recnum_t *recnum,* int *bytes_wanted)*;

DESCRIPTION

The **device_read_request** function reads a sequence of bytes from a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call does not wait for the data to be read. The data will be subsequently sent in a message to *reply_port*. See **device_read_reply**.

PARAMETERS

device
: A device port to the device to be read.

reply_port
: The port to which the reply message is to be sent.

mode
: I/O mode value. Meaningful options are:

  D_NOWAIT
  : Do not wait if data is unavailable.

recnum
: Record number to be read.

bytes_wanted
: Size of data transfer.

RETURN VALUE

KERN_SUCCESS
: Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_read_inband**, **device_read_inband_reply**, **device_read_-inband_request**, **device_read_reply**, **device_read**.

device_read_request_inband

device_read_request_inband Function — Asynchronously read a sequence of bytes "inband" from a device object.

LIBRARY

libmach.a

#include <device_request.h>

SYNOPSIS

kern_return_t **device_read_request_inband**(device_t *device,* mach_port_t *reply_port,* dev_mode_t *mode,* recnum_t *recnum,* int *bytes_wanted)*;

DESCRIPTION

The **device_read_request_inband** function reads a sequence of bytes from a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call does not wait for the data to be read. The data will be subsequently sent in a message to *reply_port*. This call differs from **device_read_request** in that the returned data bytes will be sent "inband" in the reply IPC message. See **device_read_reply_inband**.

PARAMETERS

device
: A device port to the device to be read.

reply_port
: The port to which the reply message is to be sent.

mode
: I/O mode value. Meaningful options are:

  D_NOWAIT
  : Do not wait if data is unavailable.

recnum
: Record number to be read.

bytes_wanted
: Size of data transfer.

RETURN VALUE

KERN_SUCCESS
: Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_read_inband**, **device_read_inband_reply**, **device_read_-inband**, **device_read_reply**, **device_read, device_read_request**.

device_reply_server

device_reply_server Function — Handles messages from a kernel device driver

LIBRARY

libmach.a

SYNOPSIS

boolean_t **device_reply_server**(mach_msg_header_t **in_msg,*
mach_msg_header_t **out_msg)*;

DESCRIPTION

The **device_reply_server** function is the MIG generated server handling function to handle messages from kernel device drivers. Such messages were sent in response to the various **device_..._request**... calls. It is assumed when using those calls that some task is listening for reply messages on the port named as a reply port to those calls. The **device_reply_server** function performs all necessary argument handling for a kernel message and calls one of the following functions to interpret the message:

- **device_open_reply**
- **device_write_reply**
- **device_write_reply_inband**
- **device_read_reply**
- **device_read_reply_inband**

PARAMETERS

in_msg
The device driver message received from the kernel.

out_msg
A reply message. No messages from a device driver expect a direct reply, so this field is not used.

RETURN VALUE

TRUE
The message was handled and the appropriate function was called.

FALSE
The message did not apply to this device handler interface and no other action was taken.

RELATED INFORMATION

Functions: **device_open_reply**, **device_write_reply**, **device_write_reply_inband**, **device_read_reply**, **device_read_reply_inband**.

device_set_filter

device_set_filter Function — Names an input filter for a device

LIBRARY

libmach.a

#include <device.h>

#include <device/net_status.h>

SYNOPSIS

kern_return_t **device_set_filter** (device_t *device,* mach_port_t *receive_port,* mach_msg_type_name_t *receive_port_type,* int *priority,* filter_array_t **filter,* mach_msg_type_number_t *filter_count)*;

DESCRIPTION

The **device_set_filter** function provides a means by which selected data appearing at a device interface can be selected and routed to a port.

The filter command list consists of an array of up to NET_MAX_FILTER (unsigned short) words to be applied to incoming messages to determine if those messages should be given to a particular input filter.

Each filter command list specifies a sequences of actions which leave a boolean value on the top of an internal stack. Each word of the command list specifies specifies a data operation (high order NETF_NBPO bits) as well as a binary operator (low order NETF_NBPA bits).

The value to be pushed onto the stack is chosen as follows:

NETF_PUSHLIT
: Use the next word of the filter as a value.

NETF_PUSHZERO
: Use 0 as the value.

NETF_PUSHWORD+*N*
: Use word *N* of the message as the value.

NETF_NOPUSH
: Don't push a value.

Once a value is pushed (except for the case of NETF_NOPUSH), the top two elements of the stack are popped and a binary operator applied to them (with the old top of stack as the second operand). The result of the operator is pushed on the stack. These operators are:

NETF_NOP
: Don’t pop off any values and do no operation.

NETF_EQ
: Perform an equal comparison.

NETF_LT
: Perform a less than comparison.

NETF_LE
: Perform a less than or equal comparison.

NETF_GT
: Perform a greater than comparison.

NETF_GE
: Perform a greater than or equal comparison.

NETF_AND
: Perform a bit-wise boolean AND operation.

NETF_OR
: Perform a bit-wise boolean inclusive OR operation.

NETF_XOR
: Perform a bit-wise boolean exclusive OR operation.

NETF_NEQ
: Perform a not equal comparison.

NETF_LSH
: Perform a left shift operation.

NETF_RSH
: Perform a right shift operation.

NETF_ADD
: Perform an addition.

NETF_SUB
: Perform a subtraction.

NETF_COR
: Perform an equal comparison. If the comparison is TRUE, terminate the filter list. Otherwise, pop the result of the comparison off the stack.

NETF_CAND
: Perform an equal comparison. If the comparison is FALSE, terminate the filter list. Otherwise, pop the result of the comparison off the stack.

NETF_CNOR
: Perform a not equal comparison. If the comparison is FALSE, terminate the filter list. Otherwise, pop the result of the comparison off the stack.

NETF_CNAND
: Perform a not equal comparison. If the comparison is TRUE, terminate the filter list. Otherwise, pop the result of the comparison off the stack.

The scan of the filter list terminates when the filter list is emptied, or a NETF_C... operation terminates the list. At this time, if the final value of the top of the stack is TRUE, then the message is accepted for the filter.

PARAMETERS

device
: A device port

receive_port
: The port to receive the input data that is selected by the filter.

receive_port_type
: IPC type of the send right provided to the device; either MACH_MSG_TYPE_MAKE_SEND or MACH_MSG_TYPE_MOVE_SEND.

priority
: Used to order multiple receivers.

filter
: The address of an array of filter values.

filter_count
: The size of the *filter* array.

RETURN VALUE

D_SUCCESS
: Device filter set.

D_NO_SUCH_DEVICE
: Device is not open or operational.

D_INVALID_OPERATION
: No *receive_port* was supplied.

device_set_status

device_set_status Function — Sets device status.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_set_status** (device_t *device,* int *flavor,* dev_status_t *status,* mach_msg_type_number_t *status_count)*;

DESCRIPTION

The **device_set_status** function sets device status. The possible values of *flavor* as well as the corresponding meanings are device dependent.

PARAMETERS

device
: A device port to the device to be manipulated.

flavor
: The type of status information to set.

status
: The status information to set.

status_count
: The size of the status information.

RETURN VALUE

D_SUCCESS
: Device status was set.

D_NO_SUCH_DEVICE
: The device is not open or operational.

RELATED INFORMATION

Functions: **device_get_status**.

device_write

device_write Function — Write a sequence of bytes to a device object.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_write** (device_t *device,* dev_mode_t *mode,* recnum_t *recnum,* io_buf_ptr_t *data,* mach_msg_type_number_t *data_count,* int **bytes_written)*;

DESCRIPTION

The **device_write** function writes a sequence of bytes to a device object. The meaning of *recnum* as well as the specific operation performed is device dependent.

PARAMETERS

device
: A device port to the device to be written.

mode
: I/O mode value. Meaningful options are:

  D_NOWAIT
  : Do not wait for I/O completion.

recnum
: Record number to be written.

data
: Data bytes to be written.

data_count
: Number of data bytes to be written.

bytes_written
: [out] Size of data transfer.

RETURN VALUE

D_SUCCESS
: Data was written.

D_NO_SUCH_DEVICE
: The device is dead or not completely open.

RELATED INFORMATION

Functions: **device_write_inband**, **device_write_inband_reply**, **device_-write_inband_request**, **device_write_reply**, **device_write_request**.

device_write_inband

device_write_inband Function — Write a sequence of bytes “inband” to a device object.

LIBRARY

libmach.a

#include <device.h>

SYNOPSIS

kern_return_t **device_write_inband**(device_t *device,* dev_mode_t *mode,*
recnum_t *recnum,* io_buf_ptr_inband_t *data,*
mach_msg_type_number_t *data_count,* int
**bytes_written)*;

DESCRIPTION

The **device_write** function writes a sequence of bytes to a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call differs from **device_write** in that the bytes to be written are sent “inband” in the request IPC message.

PARAMETERS

device
A device port to the device to be written.

mode
I/O mode value. Meaningful options are:

D_NOWAIT
Do not wait for I/O completion.

recnum
Record number to be written.

data
Data bytes to be written.

data_count
Number of data bytes to be written.

bytes_written
[out] Size of data transfer.

RETURN VALUE

D_SUCCESS
Data was written.

D_NO_SUCH_DEVICE
The device is dead or not completely open.

RELATED INFORMATION

Functions: **device_write**, **device_write_inband_reply**, **device_write_inband_request**, **device_write_reply**, **device_write_request**.

device_write_reply

device_write_reply Server Interface — Receives the reply message from an asynchronous write request.

SYNOPSIS

kern_return_t **device_write_reply**(mach_port_t *reply_port,* kern_return_t *return_code,* int *bytes_written)*;

DESCRIPTION

A **device_write_reply** function is called as the result of a kernel message indicating the completion status resulting from a previous **device_write_request** call.

PARAMETERS

reply_port
The port to which the reply message was sent.

return_code
The return status code from the write.

bytes_written
Number of data bytes written.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_write_inband**, **device_write_inband_reply**, **device_write_inband_request**, **device_write**, **device_write_request**.

device_write_reply_inband

device_write_reply_inband Server Interface — Receives the reply message from an asynchronous "inband" write request.

SYNOPSIS

kern_return_t **device_write_reply_inband**(mach_port_t *reply_port,*
kern_return_t *return_code,* int *bytes_written)*;

DESCRIPTION

A **device_write_reply_inband** function is called as the result of a kernel message returning the completion status resulting from a previous **device_write_request_inband** call.

PARAMETERS

reply_port
The port to which the reply message was sent.

return_code
The return status code from the write.

bytes_written
Number of data bytes written.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_write_inband**,**device_write_inband_request**, **device_write**, **device_write_reply, device_write_request**.

device_write_request

device_write_request Function — Asynchronously write a sequence of bytes to a device object.

LIBRARY

libmach.a

#include <device_request.h>

SYNOPSIS

kern_return_t **device_write_request**(device_t *device,* mach_port_t *reply_port,* dev_mode_t *mode,* recnum_t *recnum,* io_buf_ptr_t *data,* mach_msg_type_number_t *data_count)*;

DESCRIPTION

The **device_write_request** function writes a sequence of bytes to a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call does not wait for the data to be written. The result of the write will be subsequently sent in a message to *reply_port*. See **device_write_reply**.

PARAMETERS

device
: A device port to the device to be written.

reply_port
: The port to which the reply message is to be sent.

mode
: I/O mode value. Meaningful options are:

  D_NOWAIT
  : Do not wait for I/O completion.

recnum
: Record number to be written.

data
: Data bytes to write.

data_count
: Number of bytes to write.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_write_inband**, **device_write_inband_reply**, **device_-write_inband_request**, **device_write_reply**, **device_write**.

device_write_request_inband

device_write_request_inband Function — Asynchronously write a sequence of bytes "inband" to a device object.

LIBRARY

libmach.a

#include <device_request.h>

SYNOPSIS

kern_return_t **device_write_request_inband**(device_t *device,* mach_port_t *reply_port,* dev_mode_t *mode,* recnum_t *recnum,* io_buf_ptr_inband_t *data,* mach_msg_type_number_t *data_count)*;

DESCRIPTION

The **device_write_request_inband** function writes a sequence of bytes to a device object. The meaning of *recnum* as well as the specific operation performed is device dependent. This call does not wait for the data to be written. The result of the write will be subsequently sent in a message to *reply_port*. This call differs from **device_write_request** in that the data bytes to be written are sent "inband" in the request IPC message. See **device_write_reply_inband**.

PARAMETERS

device
: A device port to the device to be written.

reply_port
: The port to which the reply message is to be sent.

mode
: I/O mode value. Meaningful options are:

  D_NOWAIT
  : Do not wait for I/O completion.

recnum
: Record number to be written.

data
: Data bytes to write.

data_count
: Number of bytes to write.

RETURN VALUE

KERN_SUCCESS
Since this function does not receive a reply message, it has no return value. Only message transmission errors apply.

RELATED INFORMATION

Functions: **device_write_inband**, **device_write_inband_reply**, **device_-write_inband**, **device_write_reply**, **device_write, device_write_request**.

CHAPTER 11 Miscellaneous Interface Specifications

This chapter is a place holder for the various chapters that will define miscellaneous interfaces.

Initialization Interfaces

This section will discuss any initialization interfaces that would be available to servers

Configuration Interfaces

This section will discuss any configuration interfaces that would be available to servers or at install/build time

Network Interfaces

This section will discuss any network interfaces that would be available in the kernel. (More investigation is necessary ti identify any interfaces that would fall into this category.)

“Build-Time” Interfaces

This section will discuss any interfaces or information that would be available externally at build or install time

Security Interfaces

This section will discuss any external interfaces that are related to security in the kernel.