APPENDIX D Multicomputer Support

Support for multicomputers is being added to the Mach kernel. This provides transparent support for distributed, non-shared-memory environments. The current support does not handle node failures and so is suitable to multicomputer environments but not yet to networked workstation environments.

With this support, a single logical Mach kernel is formed that spans a set of computers. This support transparently distributes Mach IPC and virtual memory. However, each host (called a *node*) within the multicomputer maintains its identity (separate control and name ports, processor sets, devices, etc.).

This appendix describes operations that apply to individual nodes in such a configuration.

Status:

Believed to be stable

norma_get_special_port

Function — Returns a send right to a node specific port

LIBRARY

#include <**mach/norma_special_ports.h**>

SYNOPSIS

kern_return_t **norma_get_special_port**
(host_control_port_t *host_priv,*
node_t *node,*
norma_special_port_t *which_port,*
mach_port_t* *special_port)*;

norma_get_device_port

Macro form

kern_return_t **norma_get_device_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_DEVICE_PORT, *special_port*)

norma_get_host_paging_port

Macro form

kern_return_t **norma_get_host_paging_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_HOST_PAGING_PORT, *special_port*)

norma_get_host_port

Macro form

kern_return_t **norma_get_host_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_HOST_PORT, *special_port*)

norma_get_host_priv_port

Macro form

kern_return_t **norma_get_host_priv_port**
(host_control_port_t *host_priv,*

node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_HOST_PRIV_PORT, *special_port*)

norma_get_security_port

Macro form

kern_return_t **norma_get_security_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_SECURITY_PORT, *special_port*)

norma_get_nameserver_port

Macro form

kern_return_t **norma_get_nameserver_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t* *special_port)*

⇒ **norma_get_special_port** (*host_priv*, *node*, NORMA_NAMESERVER_PORT, *special_port*)

DESCRIPTION

The **norma_get_special_port** function returns a send right for a special port belonging to *node* on *host_priv*.

Each node maintains a (small) set of node specific ports. The device master port, host paging port, host name, host control, and security ports are maintained by the kernel. The kernel also permits a small set of server specified node specific ports; the name server port is an example and is given (by convention) an assigned special port index.

PARAMETERS

host_priv
[in host-control send right] The control port for the host for which to return the special port's send right.

node
[in scalar] The index of the node for which the port is desired.

which_port
[in scalar] The index of the special port for which the send right is requested. Valid values are:

NORMA_DEVICE_PORT
: [device-master send right] The device master port for the node.

NORMA_HOST_PAGING_PORT
: [default-pager send right] The default pager port for the node.

NORMA_HOST_PORT
: [host-name send right] The host name port for the node. If the specified node is the current node, this value (unless otherwise set) is the same as would be returned (by default) by **mach_host_self**.

NORMA_HOST_PRIV_PORT
: [host-control send right] The host control port for the node.

NORMA_SECURITY_PORT
: [security send right] The security port for the node.

NORMA_NAMESERVER_PORT
: [name-server send right] The registered name server port for the node.

special_port
: [out norma-special send right] The returned value for the port.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **mach_host_self**, **norma_set_special_port**, **vm_set_default_memory_manager**, **bootstrap_ports**.

norma_port_location_hint

Status:

Believed to be stable

Function — Guess a port's current location

LIBRARY

Not declared anywhere.

SYNOPSIS

```
kern_return_t norma_port_location_hint
        (task_port_t          task,
         mach_port_name_t     name,
         node_t*              node);
```

DESCRIPTION

The **norma_port_location_hint** function returns the best guess of *name*'s current location. The hint is guaranteed to be a node where the port once was; it is guaranteed to be accurate if port has never moved. This can be used to determine residence node for hosts, tasks, threads, etc.

PARAMETERS

task
: [in task send right] Task containing the right to locate

name
: [in scalar] Name of the right to locate

node
: [out scalar] Port location hint

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **norma_task_create**.

Status:

Believed to be stable [OSF-RI NMK17].

norma_node_self

Function — Return the node index of the current host

LIBRARY

Not declared anywhere.

SYNOPSIS

```
kern_return_t norma_node_self
        (host_t        host,
         node_t *      node);
```

DESCRIPTION

The **norma_node_self** function returns the node index of the current host.

PARAMETERS

host
[in host send-right] Name of the host.

node
[out scalar] Node index of the host.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **norma_task_create**, **norma_task_clone**.

norma_set_special_port

Status:

Believed to be stable

Function — Sets a node specific special port

LIBRARY

#include **<mach/norma_special_ports.h>**

SYNOPSIS

kern_return_t **norma_set_special_port**
(host_control_port_t *host_priv,*
node_t *node,*
norma_special_port_t *which_port,*
mach_port_t *special_port)*;

norma_set_device_port

Macro form

kern_return_t **norma_set_device_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_DEVICE_PORT, *special_port*)

norma_set_host_paging_port

Macro form

kern_return_t **norma_set_host_paging_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_HOST_PAGING_PORT, *special_port*)

norma_set_host_port

Macro form

kern_return_t **norma_set_host_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_HOST_PORT, *special_port*)

norma_set_host_priv_port

Macro form

kern_return_t **norma_set_host_priv_port**
(host_control_port_t *host_priv,*

node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_HOST_PRIV_PORT, *special_port*)

norma_set_security_port

Macro form

kern_return_t **norma_set_security_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_SECURITY_PORT, *special_port*)

norma_set_nameserver_port

Macro form

kern_return_t **norma_set_nameserver_port**
(host_control_port_t *host_priv,*
node_t *node*,
mach_port_t *special_port)*

⇒ **norma_set_special_port** (*host_priv*, *node*, NORMA_NAMESERVER_PORT, *special_port*)

DESCRIPTION

The **norma_set_special_port** function sets the special port belonging to *node* on *host_priv*.

Each node maintains a (small) set of node specific ports. The device master port, host paging port, host name, host control, and security ports are maintained by the kernel. The kernel also permits a small set of server specified node specific ports; the name server port is an example and is given (by convention) an assigned special port index.

PARAMETERS

host_priv
[in host-control send right] The host for which to set the special port. Currently, this must be the per-node host control port.

node
[in scalar] The index of the node for which the port is to be set.

which_port
[in scalar] The index of the special port to be set. Valid values are:

NORMA_DEVICE_PORT
: [device-master send right] The device master port for the node.

NORMA_HOST_PAGING_PORT
: [default-pager send right] The default pager port for the node.

NORMA_HOST_PORT
: [host-name send right] The host name port for the node.

NORMA_HOST_PRIV_PORT
: [host-control send right] The host control port for the node.

NORMA_SECURITY_PORT
: [security send right] The security port for the node.

NORMA_NAMESERVER_PORT
: [name-server send right] The registered name server port for the node.

special_port
: [in norma-special send right] Send right to the new special port.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **mach_host_self**, **norma_get_special_port**, **vm_set_default_memory_manager**.

Status:

Believed to be stable

norma_task_clone

Function — "Clone" a task on a specified node

LIBRARY

Not declared anywhere.

SYNOPSIS

kern_return_t **norma_task_clone**

(task_port_t	*parent_task,*
boolean_t	*inherit_memory*,
node_t	*child_node*,
task_port_t*	*child_task)*;

DESCRIPTION

The **norma_task_clone** function "clones" a new task from *parent_task* on the specified *node* and returns the name of the new task in *child_task*. The child task acquires shared parts of the parent's address space (see **vm_inherit**) regardless of the inheritance set for the parent's memory regions, although the inheritance for the child's regions will be set to that of the parent's regions. The child task initially contains no threads.

By way of comparison, tasks created by the standard **task_create** primitive are created on the same node as the parent.

Other than being created on a different node, the new task has the same properties as if created by **task_create**.

PARAMETERS

parent_task
: [in task send right] The port for the task from which to draw the child task's port rights, resource limits, and address space.

inherit_memory
: [in scalar] 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_node
: [in scalar] The node index of the node on which to create the child.

child_task
: [out task send right] The kernel-assigned port name for the new task.

NOTES

This call differs from **norma_task_create** in that the inheritance set for the parent's memory regions is ignored; the child always shares memory with the parent.

This call is intended to support process migration, where the inheritance semantics of **norma_task_create** would break migrated programs that depended upon sharing relationships remaining after migration.

This call is not a true task migration call, in that it does not migrate the port space, threads, and other non-address-space attributes of the task.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **task_create**, **norma_task_create**.

Status:

Believed to be stable

norma_task_create

Function — Create a task on a specified node

LIBRARY

Not declared anywhere.

SYNOPSIS

```
kern_return_t norma_task_create
        (task_port_t        parent_task,
         boolean_t          inherit_memory,
         node_t             child_node,
         task_port_t*       child_task);
```

DESCRIPTION

The **norma_task_create** function creates a new task from *parent_task* on the specified *node* 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.

By way of comparison, tasks created by the standard **task_create** primitive are created on the same node as the parent.

Other than being created on a different node, the new task has the same properties as if created by **task_create**.

PARAMETERS

parent_task
: [in task send right] The port for the task from which to draw the child task's port rights, resource limits, and address space.

inherit_memory
: [in scalar] 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_node
: [in scalar] The node index of the node on which to create the child.

child_task
: [out task send right] The kernel-assigned port name for the new task.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **task_create**, **norma_task_clone**.

Status:

To be modified due to new NoRMA IPC system [OSF-RI NMK17].

norma_task_teleport

Function — “Clone” a task on a specified node

LIBRARY

Not declared anywhere.

SYNOPSIS

```
kern_return_t norma_task_teleport
                (task_port_t        parent_task,
                 boolean_t          inherit_memory,
                 node_t             child_node,
                 task_port_t*       child_task);
```

DESCRIPTION

The **norma_task_clone** function “clones” a new task from *parent_task* on the specified *node* and returns the name of the new task in *child_task*. The child task acquires shared parts of the parent’s address space (see **vm_inherit**) regardless of the inheritance set for the parent’s memory regions, although the inheritance for the child’s regions will be set to that of the parent’s regions. The child task initially contains no threads. The *parent_task* is then terminated.

By way of comparison, tasks created by the standard **task_create** primitive are created on the same node as the parent.

Other than being created on a different node, the new task has the same properties as if created by **task_create**.

PARAMETERS

parent_task
: [in task send right] The port for the task from which to draw the child task’s port rights, resource limits, and address space.

inherit_memory
: [in scalar] 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_node
: [in scalar] The node index of the node on which to create the child.

child_task
: [out task send right] The kernel-assigned port name for the new task.

NOTES

This call differs from **norma_task_clone** in that the parent task is terminated as part of the **teleport** call.

This call differs from **norma_task_create** in that the inheritance set for the parent's memory regions is ignored; the child always shares memory with the parent.

This call is intended to support process migration, where the inheritance semantics of **norma_task_create** would break migrated programs that depended upon sharing relationships remaining after migration.

This call is not a true task migration call, in that it does not migrate the port space, threads, and other non-address-space attributes of the task.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **norma_task_clone**, **task_create**, **norma_task_create**.

Status:

Obsolete; to be deleted [OSF-RI NMK17].

task_set_child_node

Function — Set the node upon which future child tasks will be created

LIBRARY

Not declared anywhere.

SYNOPSIS

```
kern_return_t task_set_child_node
        (mach_port_t        task,
         int                child_node);
```

DESCRIPTION

The **task_set_child_node** function specifies a node upon which child tasks will be created. This call exists only to allow testing with unmodified servers. Server developers should use **norma_task_create** instead.

PARAMETERS

task
: [in task port] The task who's children are to be affected.

node
: [in scalar] The index of the node upon which future children should be created.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **norma_task_create**, **norma_task_clone**.

APPENDIX E Bootstrap Server Interfaces

This chapter discusses the specifics of the functions used to support a bootstrap server The control port for a host is provided to the bootstrap task for that host. 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.

NOTE that this entire chapter may be moved to a separate document in a future revision of this specification.

Status:

Believed to be stable

bootstrap_arguments

Function — Return the arguments to the bootstrap task

SYNOPSIS

kern_return_t **bootstrap_ports**	
(bootstrap_port_t	*bootstrap,*
task_port_t	*task,*
vm_offset_t*	*arguments*,
mach_msg_type_number_t*	*argumentCnt*);

DESCRIPTION

The **bootstrap_arguments** function returns a the set of arguments specified to the boot loader. The bootstrap task can act as a server on this interface for the tasks that it creates in order to pass arguments to them. This call only works against the bootstrap port given to the system bootstrap task.

The libsa_mach.a standalone Mach C runtime startup code uses this function, and **bootstrap_environment**, to initialize `argc`, `argv`, and `envp` for the entry point of tasks it creates (i.e. `main`).

PARAMETERS

bootstrap
: [in bootstrap send right] The bootstrap port of the task, obtained from **task_get_special_ports**.

task
: [in task send right] The task port for the task whose argument strings are requested.

arguments
: [out array] The byte array containing the argument strings. This is an array of *argumentCnt* bytes, containing NULL characters separating the strings.

argumentCnt
: [inout count] The number of bytes in the argument array.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **bootstrap_environment**, **bootstrap_ports**, **task_get_special_ports**.

bootstrap_environment

Status:

Believed to be stable

Function — Return the environment of the bootstrap task

SYNOPSIS

```
kern_return_t bootstrap_ports
        (bootstrap_port_t                bootstrap,
         task_port_t                     task,
         vm_offset_t*                    environment,
         mach_msg_type_number_t*         environmentCnt);
```

DESCRIPTION

The **bootstrap_environment** function returns a the set of arguments specified to the boot loader. The bootstrap task can act as a server on this interface for the tasks that it creates in order to pass arguments to them. This call only works against the bootstrap port given to the system bootstrap task.

The libsa_mach.a standalone Mach C runtime startup code uses this function, and **bootstrap_arguemnts**, to initialize `argc`, `argv`, and `envp` for the entry point of tasks it creates (i.e. `main`).

PARAMETERS

bootstrap
: [in bootstrap send right] The bootstrap port of the task, obtained from **task_get_special_ports**.

task
: [in task send right] The task port for the task whose environment is requested.

environment
: [out array] The byte array containing the environment strings. This is an array of *environmentCnt* bytes, containing NULL characters separating the strings.

environmentCnt
: [inout count] The number of bytes in the environment array.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **bootstrap_arguments**, **bootstrap_ports**, **task_get_special_ports**.

Status:

Believed to be stable

bootstrap_ports

Function — Returns the control ports for task startup

SYNOPSIS

kern_return_t **bootstrap_ports**

(bootstrap_port_t	*bootstrap,*
host_control_port_t*	*host_control,*
device_master_port_t*	*device_master*,
security_port_t*	*security*);

DESCRIPTION

The **bootstrap_ports** function returns a send right to the host control, host security and device master ports. This call only works against the bootstrap port (TASK_BOOTSTRAP_PORT) given to the system bootstrap task.

It is the responsibility of the bootstrap task to manage the distribution of these rights to other servers. An OS personality can serve as a server on the TASK_-BOOTSTRAP_PORT for tasks or servers that it manages, and can regulate or interpose on the ports in any way it deems necessary.

PARAMETERS

bootstrap
[in bootstrap send right] The bootstrap port of the bootstrap task.

host_priv
[out host-control send right] The control port for the host.

device_master
[out device-master send right] The device master port.

security
[out security send right] The host security port, used for setting task identity.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **device_open**, **host_get_clock_control**, **host_get_clock_service**, **host_processor_set_priv**, **host_processors**, **task_set_security_id**.

APPENDIX F Intel 386 Support

This appendix describes special kernel interfaces to support the special hardware features of the Intel 386 processor and its successors.

Status:

Believed to be stable

Aside from the special functions listed here, the Intel 386 support includes the following thread state "flavors" (See **mach/thread_status.h**.).

- i386_THREAD_STATE—Basic machine thread state, except for segment and floating registers.
- i386_REGS_SEGS_STATE—Same as i386_THREAD_STATE but also sets/gets segment registers.
- i386_FLOAT_STATE—Floating point registers.

The following sections also contain special state flavors pertinent to special kernel functionality.

Debug State

The 386 debug registers (0 to 3) may be set or read using the i386_DEBUG_STATE flavor. The data for this flavor is an array of structures of the following format:

```
[1]  struct i386_debug_state
[2]  {
[3]          unsigned int            address;
[4]          unsigned int            size;
[5]          unsigned int            type;
[6]  };
[7]  #define i386_DEBUG_STATE_COUNT
             (sizeof (struct i386_debug_state)/sizeof(unsigned int))
```

To set or read multiple debug registers (starting with register 0), the caller supplies a data area that is a multiple of i386_DEBUG_STATE_COUNT in size.

Coprocessor Emulation

Software emulation of the (floating point) coprocessor can be selected with the i386_COPROC_EMUL flavor. The flavor takes a value which is a single boolean. When set TRUE, all floating point instructions will raise an exception of type EXC_EMULATION (with an exception code of EXC_I386_NOEXT).

I/O Support

The 386 supports direct IO instructions. Generally speaking, these instructions are privileged (sensitive to IOPL). Mach, in combination with the processor, allows threads to directly execute these instructions against hardware IO ports for which the thread has permission (those named in its IO permission bitmap). (Note that this is a per-thread property.) The **i386_io_port_add** function enables IO to the port corresponding to the device port supplied to the call. **i386_io_port_remove** disables such IO; **i386_io_port_list** lists the devices to which IO is permitted.

For the sake of supporting the DOS emulator, the kernel supports a special device *iopl*. Access to this device implies access to the speaker, configuration CMOS, game port, sound blaster, printer and the VGA ports (device *kd0* or *vga*). Attempting to execute an IO instruction against one of these devices when the task holds send rights to the *iopl* device automatically adds these devices to the IO permission bitmap.

In the thread state→*efl* flags is a simulated interrupt request flag, EFL_IRQ. This flag can be manipulated by manipulating the i386_THREAD_STATE. As an optimization, the i386_INTERRUPT_REQUEST state flavor manipulates only this one flag, taking a boolean value as the flavor's data.

Virtual 8086 Support

Virtual 8086 mode is supported by Mach, enabled when the EFL_VM (virtual machine) flag in the thread state→*efl* is set. The various instructions sensitive to IOPL are simulated by the Mach kernel. This includes simulating an interrupt enabled flag and associated instructions.

A virtual 8086 task receives simulated 8086 interrupts by setting an interrupt descriptor table (in task space). This table is set with the i386_V86_ASSIST_STATE state flavor.

```
struct i386_v86_assist_state
{
        unsigned int                int_table;
        int                         int_count;
};
#define i386_V86_ASSIST_STATE_COUNT
            (sizeof (struct i386_v86_assist_state)/sizeof(unsigned int))
```

The *int_table* field points to an interrupt table in task space. The table has *int_count* entries. Each entry of this table has the format shown below.

```
[1]  struct v86_interrupt_table
[2]  {
[3]           unsigned int             count;
[4]           unsigned short           mask;
[5]           unsigned short           vec;
[6]  };
```

When the 8086 task has an associated interrupt table and its simulated interrupt enable flag is set (EFL_IRQ), the kernel will scan the table looking for an entry whose *count* is greater than zero and whose *mask* value is not set. If found, the count will be decremented and the task will take a simulated 8086 interrupt to the address given by *vec*. No other simulated interrupts will be generated until the 8086 task executes an *iret* instruction and the (simulated) interrupt enable flag is again set. The generation of the simulated interrupt will turn off the hardware's trace trap flag; executing the *iret* instruction will restore the trace trap flag.

The i386_INT_BITMAP state flavor takes a bitmap of 256 bits used to control software interrupts. If bit *N* is 0 then an "INT $*N*" instruction will execute as it would in real mode; otherwise such an instruction will take a general protection exception. The default value for this bitmap is all 1 bits. This bitmap only affects virtual 8086 threads.

Local Descriptor Table

Although the 386 (and successors) view the address space as segmented, Mach provides each task with a linear address space (32 bits for the Intel family). The various entries in the system global descriptor table (GDT) are used for system use; in general the entries map all of kernel memory. The thread's local descriptor table (LDT) maps its task space. Segment 2 of this table is used for task code accesses (it permits only read access); segment 3 is used for data accesses (it permits write access, subject to page level protections); both segments, though, map all of the task's address space. Segment 1 of the table is unused. Segment 0 is used as a call gate for system calls (traps).

Each thread may set entries in its LDT to describe various ranges of its underlying address space. There is no way that this mechanism permits a thread to access any more virtual memory than its address space permits; these LDT segment entries merely provide different views of the address space. A segment may be thought of as an automatically relocated portion of the address space; the beginning of a segment can be referenced as address zero given the appropriately set 386 segment register. These local segment descriptors are manipulated with the **i386_set_ldt** function and examined with the **i386_get_ldt** function. A thread may map this LDT into its address space (read-only) with the **i386_map_ldt_readonly** function.

Status:

Believed to be stable

i386_get_ldt

Function — Return per-thread segment descriptors

LIBRARY

#include <**mach/i386/mach_i386.h**>

SYNOPSIS

[1]	struct **descriptor**	
[2]	{	
[3]	unsigned int	*low_word*;
[4]	unsigned int	*high_word*;
[5]	};	
[6]	typedef struct descriptor	**descriptor_t**;
[7]	typedef struct descriptor*	**descriptor_list_t**;

kern_return_t **i386_get_ldt**

(thread_port_t	*thread,*
int	*first_selector*,
int	*desired_count*,
descriptor_list_t*	*desc_list,*
mach_msg_type_number_t*	*desc_list_count)*;

DESCRIPTION

The **i386_get_ldt** function returns per-thread segment descriptors from the thread's local descriptor table (LDT).

PARAMETERS

thread
: [in thread send right] Thread whose segment descriptors are to be returned

first_selector
: [in scalar] Selector value (segment register value) corresponding to the first segment whose descriptor is to be returned

desired_count
: [in scalar] Number of returned descriptors desired

desc_list
: [out pointer to dynamic array of *descriptor_t*] Array of segment descriptors.

desc_list_count
[in/out scalar] On input, the maximum size of the buffer; on output, the size returned (in natural-sized units).

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_set_ldt**, **i386_map_ldt_readonly**.

Status:

Believed to be stable

i386_io_port_add

Function — Permit IO instructions to be performed against a device

LIBRARY

#include <**mach/i386/mach_i386.h**>

SYNOPSIS

kern_return_t **i386_io_port_add**
(thread_port_t *thread,*
device_port_t *device)*;

DESCRIPTION

The **i386_io_port_add** function adds a device to the IO permission bitmap for a thread, thereby permitting the thread to execute IO instructions against the device.

PARAMETERS

thread
[in thread send right] Thread whose permission bitmap is to be set.

device
[in device send right] The device to which IO instructions are to be permitted.

NOTES

Normally, the thread must have called **i386_io_port_add** for all devices to which it will execute IO instructions. However, possessing send rights to the *iopl* device port will cause the *iopl* device to be automatically added to the thread's IO map upon first attempted access. This is a backward compatibility feature for the DOS emulator.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_io_port_list**, **i386_io_port_remove**.

i386_io_port_list

Status:

Believed to be stable

Function — List devices permitting IO

LIBRARY

#include **<mach/i386/mach_i386.h>**

SYNOPSIS

kern_return_t **i386_io_port_list**
(thread_port_t *thread,*
device_port_array_t* *list*,
mach_msg_type_number_t* *count)*;

DESCRIPTION

The **i386_io_port_list** function returns a list of the devices named in the thread's IO permission bitmap, namely those permitting IO instructions to be executed against them.

PARAMETERS

thread
: [in thread send right] Thread whose permission list is to be returned

list
: [out pointer to dynamic array of device send rights] Device ports permitting IO

count
: [out scalar] The number of ports returned

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_io_port_add**, **i386_io_port_remove**.

Status:

Believed to be stable

i386_io_port_remove

Function — Disable IO instructions against a device

LIBRARY

#include <**mach/i386/mach_i386.h**>

SYNOPSIS

```
kern_return_t i386_io_port_remove
                (thread_port_t          thread,
                 device_port_t          device);
```

DESCRIPTION

The **i386_io_port_remove** function removes the specified device from the thread's IO permission bitmap, thereby prohibiting IO instructions being executed against the device.

PARAMETERS

thread
: [in thread send right] Thread whose permission bitmap is to be cleared

device
: [in device send right] Device whose permission is to be revoked

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_io_port_add**, **i386_io_port_list**.

i386_map_ldt_readonly

Status:

Believed to be stable

Function — Directly maps (read-only) a thread's LDT into the task's address space

LIBRARY

#include **<mach/i386/mach_i386.h>**

SYNOPSIS

kern_return_t **i386_map_ldt_readonly**

(thread_port_t	*thread,*
vm_address_t	*address*,
vm_size_t	*size)*;

DESCRIPTION

The **i386_map_ldt_readonly** function maps the local descriptor table of the target thread into the target threads address space, in the region designated by *address* and *size*. The mapping is read-only: changes made by **i386_set_ldt** or by the hardware (while the thread is running) are visible immediately but the region is not writable.

PARAMETERS

thread
: [in thread send right] Thread whose LDT is to be mapped

address
: [in scalar] The address of the region where the LDT is to be mapped. This must correspond to a page address.

size
: [in scalar] The size (in bytes) of the region into which the LDT is to be mapped.

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_get_ldt**, **i386_set_ldt**.

Status:

Believed to be stable

i386_set_ldt

Function — Set per-thread segment descriptors

LIBRARY

#include <**mach/i386/mach_i386.h**>

SYNOPSIS

```
[1]  struct descriptor
[2]  {
[3]          unsigned int                    low_word;
[4]          unsigned int                    high_word;
[5]  };
[6]  typedef struct descriptor               descriptor_t;
[7]  typedef struct descriptor*              descriptor_list_t;
```

kern_return_t **i386_set_ldt**

```
        (thread_port_t                       thread,
         int                                 first_selector,
         descriptor_list_t                   desc_list,
         mach_msg_type_number_t              count);
```

DESCRIPTION

The **i386_set_ldt** function allows a thread to have a private local descriptor table (LDT) which allows its local segments to map various ranges of its address space.

PARAMETERS

thread

[in thread send right] Thread whose segment descriptors are to be set

first_selector

[in scalar] Selector value (segment register value) corresponding to the first segment whose descriptor is to be set

desc_list

[pointer to in array of *descriptor_t*] Array of segment descriptors. The following forms are permitted:

- Empty descriptor. The ACC_P flag (segment present) may or may not be set.
- ACC_CALL_GATE — Converted into a system call gate. The ACC_P flag must be set.

All other descriptors must have both the ACC_P flag set and specify user mode access (ACC_PL_U).

- ACC_DATA
- ACC_DATA_W
- ACC_DATA_E
- ACC_DATA_EW
- ACC_CODE
- ACC_CODE_R
- ACC_CODE_C
- ACC_CODE_CR
- ACC_CALL_GATE_16
- ACC_CALL_GATE

count

[in scalar] Number of descriptors to be set

RETURN VALUE

Only generic errors apply.

RELATED INFORMATION

Functions: **i386_get_ldt**, **i386_map_ldt_readonly**.

APPENDIX G

Data Structures

This appendix discusses the specifics of the various structures used as a part of the kernel's various interfaces. This appendix does not discuss all of the various data types used by the kernel's interfaces, only the fields of the various structures used.

Status:

New – added in support of expanded DMMI suite. **[OSF/1 R1.3]**

backing_store_basic_info

Structure — Defines information concerning backing storage

SYNOPSIS

```
struct backing_store_basic_info
{
        integer_t       pageout_calls;
        integer_t       pagein_calls;
        integer_t       pages_in;
        integer_t       pages_out;
        integer_t       pages_unavail;
        integer_t       pages_init;
        integer_t       pages_init_writes;
        integer_t       bs_pages_total;
        integer_t       bs_pages_free;
        integer_t       bs_pages_in;
        integer_t       bs_pages_in_fail;
        integer_t       bs_pages_out;
        integer_t       bs_pages_out_fail;
        integer_t       bs_priority;
        integer_t       bs_clsize;
} backing_store_basic_info_t;
```

DESCRIPTION

The **backing_store_basic_info** structure defines information concerning the default memory manager's backing storage.

FIELDS

pageout_calls
Total number of pageout calls made.

pagein_calls
Total number of pagein calls made.

pages_in
Total number of pages that have been paged in.

pages_out
Total number of pages that have been paged out.

pages_unavail
Total number of zero-fill pages.

pages_init
Total number of page init requests.

pages_init_writes
Total number of page init writes.

bs_pages_total
The total number of pages on the specified backing store.

bs_pages_free
The total number of unallocated pages on the specified backing store.

bs_pages_in
The number of page read requests handled by the specified backing store.

bs_pages_in_fail
The number of page read requests handled by the specified backing store that have failed.

bs_pages_out
The number of page write requests handled by the specified backing store.

bs_pages_out_fail
The number of page write requests handled by the specified backing store that have failed.

bs_priority
The scheduling priority for the backing store thread(s). This value is supplied in the **default_pager_backing_store_create** call.

bs_clsize
The backing store cluster size, in bytes. This value is supplied in the **default_pager_backing_store_create** call.

NOTES

This structure is machine word length specific because of the memory size returned.

RELATED INFORMATION

Functions: **default_pager_backing_store_info**, **default_pager_backing_store_create.**

Status:

Believed to be stable

default_pager_info

Structure — Defines information concerning temporary backing store

SYNOPSIS

```
[1]  struct default_pager_info
[2]  {
[3]          vm_size_t                    dpi_total_space;
[4]          vm_size_t                    dpi_free_space;
[5]          vm_size_t                    dpi_page_size;
[6]  } default_pager_info_t;
```

DESCRIPTION

The **default_pager_info** structure defines information concerning the default memory manager's backing storage consumption.

FIELDS

dpi_total_space
Total space, in bytes, allocated for default memory backing storage

dpi_free_space
Free space, in bytes, in the default manager's currently allocated paging partitions

dpi_page_size
Default manager's VM page size

NOTES

This structure is machine word length specific because of the memory size returned.

RELATED INFORMATION

Functions: **default_pager_info.**

host_basic_info

Status:

Believed to be stable

Structure — Defines basic information about a host

SYNOPSIS

```
[1] struct host_basic_info
[2] {
[3]         cpu_number_t            max_cpus;
[4]         cpu_number_t            avail_cpus;
[5]         vm_size_t               memory_size;
[6]         cpu_type_t              cpu_type;
[7]         cpu_subtype_t           cpu_subtype;
[8] };
[9] typedef struct host_basic_info*     host_basic_info_t;
```

DESCRIPTION

The **host_basic_info** structure defines the basic information available about a host.

FIELDS

max_cpus
: Maximum possible CPUs for which kernel is configured

avail_cpus
: Number of CPUs now available

memory_size
: Size of memory, in bytes

cpu_type
: CPU type

cpu_subtype
: CPU sub-type

NOTES

This structure is machine word length specific because of the memory size returned.

RELATED INFORMATION

Functions: **host_info.**

Data structures: **host_load_info**, **host_sched_info.**

Status:

Believed to be stable

host_load_info

Structure — Defines load information about a host

SYNOPSIS

[1] #define **CPU_STATE_USER** 0
[2] #define **CPU_STATE_SYSTEM** 1
[3] #define **CPU_STATE_IDLE** 2
[4] struct **host_load_info**
[5] {
[6] load_t *avenrun*[3];
[7] load_t *mach_factor*[3];
[8] };
[9] typedef struct host_load_info* **host_load_info_t**;

DESCRIPTION

The **host_load_info** structure defines the loading information available about a host. The information returned is exponential averages over three periods of time: 5, 30 and 60 seconds.

FIELDS

avenrun
load average—average number of runnable processes divided by number of CPUs

mach_factor
The processing resources available to a new thread—the number of CPUs divided by (1 + the number of threads)

RELATED INFORMATION

Functions: **host_statistics.**

Data structures: **host_basic_info**, **host_sched_info.**

host_sched_info

Status:

Believed to be stable

Structure — Defines scheduling information about a host

SYNOPSIS

```
[1]  struct host_sched_info
[2]  {
[3]          mach_msg_timeout_t          min_timeout;
[4]          mach_msg_timeout_t          min_quantum;
[5]  };
[6]  typedef struct host_sched_info*     host_sched_info_t;
```

DESCRIPTION

The **host_sched_info** structure defines the limiting scheduling information available about a host.

FIELDS

min_timeout
: Minimum time-out, in milliseconds

min_quantum
: Minimum quantum (period for which a thread can be scheduled if uninterrupted), in milliseconds

RELATED INFORMATION

Functions: **host_info.**

Data structures: **host_basic_info**, **host_load_info.**

Status:

New: [OSF-RI MK6] Added for asynchronous I/O completion status

io_done_result

Structure — Defines asynchronous I/O completion status

SYNOPSIS

```
[1]  struct io_done_result
[2]  {
[3]  struct control_info
[4]  {
[5]          int                       qd_type;
[6]          mach_port_t               qd_reqid;
[7]          kern_return_t             qd_code;
[8]          io_buf_ptr_t              qd_data;
[9]          int                       qd_count;
[10] } qd_control;
[11] io_buf_ptr_inband_t                qd_inline;
[12] };
[13] typedef struct io_done_result      io_done_result_t;
```

DESCRIPTION

The **io_done_result** structure defines the set of information made available by a device driver upon completion of an outstanding asynchronous I/O request.

FIELDS

qd_type
: Identifies the type of I/O operation that was requested. Valid values are:

IO_DONE_READ The operation was a device_read_async or device_-read_async_inband.

IO_DONE_OVERWRITE The operation was a device_read_over-write_async.

IO_DONE_WRITE The operation was a device_write_async or de-vice_write_async_inband.

qd_reqid
: An opaque data item used to uniquely identify the I/O request to the task. It is typically supplied by the task as an argument to the asychronous device request. The µKernel transparently returns this data item to the task.

qd_code
: The return code that would have been returned by the non-asynchronous variant of the I/O call (i.e. device_read for device_read_async). If

the field is set to KERN_SUCCESS, then the *qd_data* and *qd_count* fields are valid.

qd_data
: If the qd_type field is set to IO_DONE_READ or IO_DONE_OVERWRITE, then this field is a pointer to the data; otherwise, this field's contents is undefined.

qd_count
: The number of bytes read or written.

qd_inline
: Contains the actual data, pointed to by *qd_data*, used in **device_read_async_inband**. In all other cases, this field is undefined.

RELATED INFORMATION

Functions: **io_done_queue_create**, **io_done_queue_wait**, **device_read_async**, **device_read_async_inband**, **device_read_overwrite_async**, **device_write_async**, **device_write_async_inband**.

Status:

Experimental

kernel_resource_sizes

Structure — Defines the sizes of significant kernel structures

SYNOPSIS

```
[1]  struct kernel_resource_sizes
[2]  {
[3]          ledger_item_t                    task;
[4]          ledger_item_t                    thread;
[5]          ledger_item_t                    port;
[6]          ledger_item_t                    memory_region;
[7]          ledger_item_t                    memory_object;
[8]  };
[9]  typedef struct kernel_resource_sizes*    kernel_resource_sizes_t;
```

DESCRIPTION

The **kernel_resource_sizes** structure defines the sizes of significant kernel structures. This information is of use when establishing ledger resource limits.

FIELDS

task
Space consumed by an empty task.

thread
Space consumed by a thread.

port
Space consumed by a port with an empty message queue.

memory_region
Space consumed by each distinct memory region (as reported by **vm_region**) in a task's address space.

memory_object
Space consumed to manage a memory object with no resident pages or copy objects.

RELATED INFORMATION

Functions: **host_info**, **ledger_create**.

mach_msg_descriptor

Status:

Believed to be stable

Structure — Defines the disposition of kernel processed message data

SYNOPSIS

```
typedef struct
{
        void*                           pad1;
        mach_msg_size_t                 pad2;
        unsigned int                    pad3: 24;
        mach_msg_descriptor_type_t      type: 8;
} mach_msg_type_descriptor_t;
typedef struct
{
        mach_port_t                     name;
        mach_msg_size_t                 pad1;
        unsigned int                    pad2: 16;
        mach_msg_type_name_t            disposition: 8;
        mach_msg_descriptor_type_t      type: 8;
} mach_msg_port_descriptor_t;
typedef struct
{
        void*                           address;
        mach_msg_size_t                 size;
        boolean_t                       deallocate: 8;
        mach_msg_copy_options_t         copy: 8;
        unsigned int                    pad1: 8;
        mach_msg_descriptor_type_t      type: 8;
} mach_msg_ool_descriptor_t;
typedef struct
{
        void*                           address;
        mach_msg_size_t                 count;
        boolean_t                       deallocate: 8;
        mach_msg_copy_options_t         copy: 8;
        mach_msg_type_name_t            disposition: 8;
        mach_msg_descriptor_type_t      type: 8;
} mach_msg_ool_ports_descriptor_t;
typedef union
{
        mach_msg_type_descriptor_t      type;
        mach_msg_port_descriptor_t      port;
        mach_msg_ool_descriptor_t       out_of_line;
        mach_msg_ool_ports_descriptor_t ool_ports;
} mach_msg_descriptor_t;
```

DESCRIPTION

A **mach_msg_descriptor** structure describes the processing to be performed for an element of kernel processed data in a Mach message.

FIELDS

name
: For single port descriptors, the name of the port whose right is being sent.

disposition
: For single port or out-of-line port array descriptors, the IPC processing to be done for the rights for the named ports.

address
: For out-of-line data or port array descriptors, the address of the out-of-line data or port (name) array.

size
: For out-of-line data descriptors, the size of the out-of-line region, in bytes.

deallocate
: For out-of-line data descriptors, true if the set of pages containing the array should be de-allocated when the message is sent.

copy
: For out-of-line descriptors, a description of the method by which the data should be copied.

count
: For out-of-line port array descriptors, the number of port names in the array.

type
: For any type of descriptor, the type of descriptor.

port
: A descriptor that describes a single port right.

out_of_line
: A descriptor that describes an out-of-line data array.

ool_ports
: A descriptor that describes an out-of-line port array.

RELATED INFORMATION

Functions: **mach_msg**.

Data structures: **mach_msg_header**.

Status:

Believed to be stable

mach_msg_header

Structure — Defines the header portion for messages

SYNOPSIS

```
typedef struct
{
        mach_msg_bits_t            msgh_bits;
        mach_msg_size_t            msgh_size;
        mach_port_t                msgh_remote_port;
        mach_port_t                msgh_local_port;
        mach_msg_size_t            msgh_reserved;
        mach_msg_id_t              msgh_id;
} mach_msg_header_t;
typedef struct
{
        mach_msg_size_t            msgh_descriptor_count;
} mach_msg_body_t;
typedef struct
{
        mach_msg_trailer_type_t    msgh_trailer_type;
        mach_msg_trailer_size_t    msgh_trailer_size;
} mach_msg_trailer_t;
typedef struct
{
        mach_msg_trailer_type_t    msgh_trailer_type;
        mach_msg_trailer_size_t    msgh_trailer_size;
        mach_port_seqno_t          msgh_seqno;
} mach_msg_seqno_trailer_t;
typedef struct
{
        mach_msg_trailer_type_t    msgh_trailer_type;
        mach_msg_trailer_size_t    msgh_trailer_size;
        mach_port_seqno_t          msgh_seqno;
        security_id_t              msgh_sender;
} mach_msg_security_trailer_t;
```

DESCRIPTION

The **mach_msg_header** structure defines the fixed size header of a Mach message. The header is followed by a message body containing data and port descriptors and zero or more data bytes.

If the MACH_MSGH_BITS_COMPLEX flag in the *msgh_bits* field is not set, then this is a simple message described by **mach_msg_header_t**. In this case, the header is immediately followed by un-typed data.

If the complex flag is set, then this is a "complex" message consisting of a **mach_msg_header_t** structure followed by a **mach_msg_body_t** structure containing a count followed by an array of descriptors specifying the disposition (processing) to be performed for the out-of-line memory regions and additional port rights.

Following the header (and any kernel processed descriptors), at natural alignment can be additional (un-typed) data, up to the size of the message (*msgh_size*). This extra data typically carries information used to decode the data stream and out-of-line data.

At the next natural boundary following the message data is the message trailer (**mach_msg_trailer_t**). This structure indicates the type and length of the trailer. If the length is greater than sizeof (**mach_msg_trailer_t**), additional fields follow providing kernel generated message attributes.

FIELDS

msgh_bits
: This field specifies the following properties of the message:

MACH_MSGH_BITS_REMOTE_MASK
: Encodes *mach_msg_type_name_t* values that specify the port rights in the *msgh_remote_port* field. The value must specify a send or send-once right for the destination of the message.

MACH_MSGH_BITS_LOCAL_MASK
: Encodes *mach_msg_type_name_t* values that specify the port rights in the *msgh_local_port* field. If the value doesn't specify a send or send-once right for the message's reply port, it must be zero and *msgh_local_port* must be MACH_PORT_NULL.

MACH_MSGH_BITS_COMPLEX
: The complex bit must be specified if the message body contains additional port rights or out-of-line memory regions.

MACH_MSGH_BITS_REMOTE(*bits*)
: This macro returns the appropriate *mach_msg_type_name_t* values, given a *msgh_bits* value.

MACH_MSGH_BITS_LOCAL(*bits*)
: This macro returns the appropriate *mach_msg_type_name_t* values, given a *msgh_bits* value.

MACH_MSGH_BITS (*remote*, *local*)
: This macro constructs a value for *msgh_bits*, given two *mach_msg_type_name_t* values.

msgh_size

This field is ignored on send (the size to send is specified by the *send_-size* parameter to **mach_msg**); the field is set on receive to the sum of the message header and body sizes (in bytes). Note that this value may be different from the send size specified by the sender if the sender and receiver machines have differing sizes for port names, memory addresses or memory range sizes.

msgh_remote_port

When sending, specifies the destination port of the message. The field must carry a legitimate send or send-once right for a port. When received, this field is swapped with *msgh_local_port*.

msgh_local_port

When sending, specifies an auxiliary port right, which is conventionally used as a reply port by the recipient of the message. The field must carry a send right, a send-once right, MACH_PORT_NULL, or MACH_PORT_DEAD. When received, this field is swapped with *msgh_remote_port*.

msgh_id

Not set or read by the **mach_msg** call. The conventional meaning is to convey an operation or function ID.

msgh_descriptor_count

The number of descriptors of kernel processed data (port rights and out-of-line data).

msgh_trailer_type

An identifier of the trailer version. Different values indicate not necessarily compatible trailer formats. The current (and only) trailer format is MACH_MSG_TRAILER_FORMAT_0. There is currently only one attribute defined within this trailer type: the sender's identity.

msgh_trailer_size

The length, in bytes, of the message trailer, including the trailer type and length fields.

msgh_seqno

The sequence number of this message relative to the port from which it is received.

msgh_sender

The security ID of the sender of the message.

RELATED INFORMATION

Functions: **mach_msg**.

Data structures: **mach_msg_descriptor**.

Status:

Believed to be stable, except for the size_limit **ledger item**

mach_port_limits

Structure — Defines queue and resource limits for a port

SYNOPSIS

```
[1] struct mach_port_limits
[2] {
[3]         mach_port_msgcount_t        queue_limit;
[4]         mach_port_oolcount_t        memory_limit;
[5]         mach_port_rightcount_t      right_limit;
[6]         ledger_item_t               size_limit;
[7] };
[8] typedef struct mach_port_limits*    mach_port_limits_t;
```

DESCRIPTION

The **mach_port_limits** structure defines various limits governing the messages that can be sent through the port.

FIELDS

queue_limit
: Number of messages allowed to be on the message queue at any given time. Attempts to queue more messages than this limit will block.

memory_limit
: Limit on the number of bytes of out-of-line data that may be received in a message.

right_limit
: Limit on the number of port rights (including the reply port) that may be received in a message.

size_limit
: Limit on the total size (in-line data, port structures, etc.) that may be consumed by a message queued on the port.

RELATED INFORMATION

Functions: **mach_port_get_attributes**, **mach_port_set_attributes**.

mach_port_status

Status:

Believed to be stable.

Changed mps_srights to an integer and listed potential values. **[OSF-RI DIPC]**

Structure — Defines information for a port

SYNOPSIS

```
[1]  struct mach_port_status
[2]  {
[3]          mach_port_name_t          mps_pset;
[4]          mach_port_seqno_t         mps_seqno;
[5]          mach_port_mscount_t       mps_mscount;
[6]          mach_port_msgcount_t      mps_msgcount;
[7]          mach_port_rights_t        mps_sorights;
[8]          integer_t                 mps_srights;
[9]          boolean_t                 mps_nsrequest;
[10] };
[11] typedef struct mach_port_status   mach_port_status_t;
```

DESCRIPTION

The **mach_port_status** structure defines information about a port.

FIELDS

mps_pset
Containing port set.

mps_seqno
Current sequence number for the port.

mps_mscount
Make-send count.

mps_msgcount
Number in the queue.

mps_sorights
How many send-once rights.

mps_srights
Indicates whether the port has send rights. Valid values are:

MPS_FALSE
The port does not have send rights.

MPS_TRUE
The port has send rights (with no more senders notification enabled).

MPS_UNKNOWN
The port does not permit no more senders notification indicating that it is unknown whether send rights exist.

mps_nsrequest
True if no-senders requested.

NOTES

This structure is machine word length sensitive due to the presence of the port set name.

RELATED INFORMATION

Functions: **mach_port_get_attributes.**

mapped_tvalspec

Status:

Believed to be stable

Structure — Defines format of kernel maintained time in a mapped clock

SYNOPSIS

```
struct mapped_tvalspec
{
        tvalspec_t              mtv_time;
        nsec_t                  mtv_csec;
};
typedef struct mapped_tvalspec  mapped_tvalspec_t;
```

DESCRIPTION

The **mapped_tvalspec** structure defines the format of the current-time structure maintained by the kernel and visible through a mapped clock (**clock_map_time**). The data in this structure is updated at the clock's current resolution and contains the same tvalspec value that would be returned by **clock_get_time**.

FIELDS

mtv_time
Clock time.

mtv_csec
A field used to synchronize with the kernel's setting of the time.

NOTES

Because of the race between the referencing of the multiple fields in the clock value and the kernel's setting them, they should be referenced as follows:

```
tvalspec_t*                     ts;
do
{
        ts → tv_sec = mtime → mtv_time.tv_sec;
        ts → tv_nsec = mtime → mtv_time.tv_nsec;
} while (ts → tv_sec!= mtime → mtv_csec);
```

RELATED INFORMATION

Functions: **clock_map_time**, **clock_get_time.**

Data Structures: **tvalspec**.

memory_object_attr_info

Status:

Believed to be stable

Structure — Defines the behavioral attributes for a memory object

SYNOPSIS

```
[1] struct memory_object_attr_info
[2] {
[3]         memory_object_copy_strategy_t    copy_strategy;
[4]         vm_offset_t                      cluster_size;
[5]         boolean_t                        may_cache_object;
[6]         boolean_t                        temporary;
[7]         boolean_t                        write_completions;
[8] };
[9] typedef struct memory_object_attr_info*  memory_object_attr_info_t;
```

DESCRIPTION

The **memory_object_attr_info** structure defines behavior and performance relevant memory object attributes.

FIELDS

copy_strategy
How the kernel should handle copying of regions associated with the memory object. The copy strategy cannot be changed once an object is initialized. Valid values are:

MEMORY_OBJECT_COPY_NONE
Use normal procedure when copying the memory object's data. Normally, the kernel requests each page with read access, copies the data, and then (optionally) flushes the data.

MEMORY_OBJECT_COPY_DELAY
Use copy-on-write technique. This strategy allows the kernel to efficiently copy large amounts of data and guarantees that the memory manager will not externally modify the data. It is the most commonly used copy strategy.

cluster_size
Preferred cluster size (in bytes) for the memory object. This helps to determine how many pages are transferred in individual data request and return messages.

may_cache_object
Cache indicator. If true, the kernel can cache data associated with the memory object (keep the memory object active) even if no virtual memory references to it remain.

temporary

When the last mapping of the memory object is removed, the object is destroyed without returning any in-memory pages.

write_completions

If true, the memory manager must explicitly indicate when it has disposed of returned pages with **memory_object_data_return_completed**. This attribute can only be set (implicitly) by using the privileged **memory_object_establish_priv**.

NOTES

Sharing cached data among all the clients of a memory object can have a major impact on performance, especially if it can be extended across successive, as well as concurrent, uses. For example, the memory objects that represent program images can be used regularly by different programs. By retaining the data for these memory objects in cache, the number of secondary storage accesses can be reduced significantly.

RELATED INFORMATION

Functions: **memory_object_establish**, **memory_object_get_attributes**, **memory_object_change_attributes**, **vm_region, memory_object_synchronize, memory_object_data_return_completed**, **vm_set_default_memory_manager**, **memory_object_establish_priv**, **vm_msync**.

Status:

Believed to be stable

policy_fifo_info

Structure — Defines first-in-first-out scheduling information

SYNOPSIS

```
[1] struct policy_fifo_limit
[2] {
[3]         priority_t              max_priority;
[4] };
[1] struct policy_fifo_base
[2] {
[3]         priority_t              base_priority;
[4] };
[1] struct policy_fifo_info
[2] {
[3]         priority_t              max_priority;
[4]         priority_t              base_priority;
[5]         boolean_t               depressed;
[6]         priority_t              depress_priority;
[7] };
```

DESCRIPTION

The **policy_fifo_info** structure defines the first-in-first-out scheduling policy information.

FIFO threads have two priorities associated with them by the system:

- 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 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.

FIELDS

max_priority
Maximum scheduling priority

base_priority
Scheduling priority

depressed
True if scheduling priority is depressed

depress_priority
Scheduling priority from which depressed

RELATED INFORMATION

Functions: **thread_info**, **task_info**, **processor_set_info**, **processor_set_policy_-default**, **processor_set_policy_disable**, **processor_set_policy_enable**, **processor_set_policy_limit**, **task_policy**, **thread_policy**, **thread_set_policy**.

Data Structures: **policy_rr_info**, **policy_timeshare_info**.

Status:

Believed to be stable

policy_rr_info

Structure — Defines round-robin scheduling information

SYNOPSIS

```
struct policy_rr_limit
{
        priority_t              max_priority;
};
struct policy_rr_base
{
        priority_t              base_priority;
        mach_msg_timeout_t      quantum;
};
struct policy_rr_info
{
        priority_t              max_priority;
        priority_t              base_priority;
        mach_msg_timeout_t      quantum;
        boolean_t               depressed;
        priority_t              depress_priority;
};
```

DESCRIPTION

The **policy_rr_info** structure defines the round-robin scheduling policy information.

Round-robin threads have two priorities associated with them by the system:

- 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 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.

FIELDS

max_priority
Maximum scheduling priority

base_priority
Scheduling priority

quantum
Scheduling quantum (in milliseconds)

depressed
True if scheduling priority is depressed

depress_priority
Scheduling priority from which depressed

RELATED INFORMATION

Functions: **thread_info**, **task_info**, **processor_set_info**, **processor_set_policy_default**, **processor_set_policy_disable**, **processor_set_policy_enable**, **processor_set_policy_limit**, **task_policy**, **thread_policy**, **thread_set_policy**.

Data Structures: **policy_fifo_info**, **policy_timeshare_info**.

Status:

Believed to be stable

policy_timeshare_info

Structure — Defines timeshare scheduling information

SYNOPSIS

```
struct policy_timeshare_limit
{
        priority_t                  max_priority;
};
struct policy_timeshare_base
{
        priority_t                  base_priority;
};
struct policy_timeshare_info
{
        priority_t                  max_priority;
        priority_t                  base_priority;
        priority_t                  cur_priority;
        boolean_t                   depressed;
        priority_t                  depress_priority;
};
```

DESCRIPTION

The **policy_timeshare_info** structure defines the timeshare scheduling policy information.

Timeshare threads have three priorities associated with them by the system:

- 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 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 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).

FIELDS

max_priority
: Maximum scheduling priority

base_priority
: Base scheduling priority

cur_priority
: Current scheduling priority

depressed
: True if scheduling priority is depressed

depress_priority
: Scheduling priority from which depressed

RELATED INFORMATION

Functions: **thread_info**, **task_info**, **processor_set_info**, **processor_set_policy_-default**, **processor_set_policy_disable**, **processor_set_policy_enable**, **processor_set_policy_limit**, **task_policy**, **thread_policy**, **thread_set_policy**.

Data Structures: **policy_fifo_info**, **policy_rr_info**.

Status:

Believed to be stable

processor_basic_info

Structure — Defines the basic information about a processor.

SYNOPSIS

```
[1] struct processor_basic_info
[2] {
[3]         cpu_type_t          cpu_type;
[4]         cpu_subtype_t       cpu_subtype;
[5]         boolean_t           running;
[6]         cpu_number_t        slot_num;
[7]         boolean_t           is_master;
[8] };
[9] typedef struct processor_basic_info*   processor_basic_info_t;
```

DESCRIPTION

The **processor_basic_info** structure defines the information available about a processor slot.

FIELDS

cpu_type
Type of CPU

cpu_subtype
Sub-type of CPU

running
True if the CPU is running

slot_num
Slot number of the CPU

is_master
True if this is the master processor

RELATED INFORMATION

Functions: **processor_info**.

processor_set_basic_info

Status:

Believed to be stable

Structure — Defines the basic information about a processor set.

SYNOPSIS

```
[1]  struct processor_set_basic_info
[2]  {
[3]          cpu_number_t                        processor_count;
[4]          policy_t                            default_policy;
[5]  };
[6]  typedef struct processor_set_basic_info*    processor_set_basic_info_t;
```

DESCRIPTION

The **processor_set_basic_info** structure defines the basic information available about a processor set.

FIELDS

processor_count
: Number of processors in this set

default_policy
: Default policy to assign to threads whose otherwise assigned policy is not enabled

RELATED INFORMATION

Functions: **processor_set_info**.

Data Structures: **processor_set_load_info**.

Status:

Believed to be stable

processor_set_load_info

Structure — Defines the scheduling statistics for a processor set.

SYNOPSIS

```
[1] struct processor_set_load_info
[2] {
[3]         mach_msg_type_number_t          task_count;
[4]         mach_msg_type_number_t          thread_count;
[5]         load_t                          load_average;
[6]         load_t                          mach_factor;
[7] };
[8] typedef struct processor_set_load_info*   processor_set_load_info_t;
```

DESCRIPTION

The **processor_set_load_info** structure defines the scheduling statistics maintained for a processor set.

FIELDS

task_count
: Number of tasks currently assigned to this processor set

thread_count
: Number of threads currently assigned to this processor set

load_average
: Average number of runnable processes divided by number of CPUs

mach_factor
: The processing resources available to a new thread—the number of CPUs divided by (1 + the number of threads)

RELATED INFORMATION

Functions: **processor_set_statistics**.

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

task_basic_info

Status:

Believed to be stable

Structure — Defines basic information for tasks

SYNOPSIS

```
struct task_basic_info
{
        suspend_t           suspend_count;
        vm_size_t           virtual_size;
        vm_size_t           resident_size;
        tvalspec_t          user_time;
        tvalspec_t          system_time;
        policy_t            policy;
};
typedef struct task_basic_info*     task_basic_info_t;
```

DESCRIPTION

The **task_basic_info** structure defines the basic information array for tasks. The **task_info** function returns this array for a specified task.

FIELDS

suspend_count
The current suspend count for the task.

virtual_size
The number of virtual pages for the task.

resident_size
The number of resident pages for the task

user_time
The total user run time for terminated threads within the task.

system_time
The total system run time for terminated threads within the task.

policy
Default scheduling policy to apply to new threads.

NOTES

This structure is machine word length sensitive due to the presence of the virtual address sizes.

RELATED INFORMATION

Functions: **task_info**.

Data Structures: **task_thread_times_info**, **policy_fifo_info**, **policy_rr_info**, **policy_timeshare_info**.

task_thread_times_info

Status:

Believed to be stable

Structure — Defines thread execution times information for tasks

SYNOPSIS

```
[1] struct task_thread_times_info
[2] {
[3]         tvalspec_t                          user_time;
[4]         tvalspec_t                          system_time;
[5] };
[6] typedef struct task_thread_times_info*  task_thread_times_info_t;
```

DESCRIPTION

The **task_thread_times_info** structure defines thread execution time statistics for tasks. The **task_info** function returns these times for a specified task. The **thread_info** function returns this information for a specific thread.

FIELDS

user_time
: Total user run time for live threads.

system_time
: Total system run time for live threads.

RELATED INFORMATION

Functions: **task_info**.

Data Structures: **task_basic_info, thread_info**.

Status:

Believed to be stable

thread_basic_info

Structure — Defines basic information for threads

SYNOPSIS

[1]	struct **thread_basic_info**		
[2]	{		
[3]		tvalspec_t	*user_time*;
[4]		tvalspec_t	*system_time*;
[5]		cpu_usage_t	*cpu_usage*;
[6]		policy_t	*policy*;
[7]		run_state_t	*run_state*;
[8]		thread_flags_t	*flags*;
[9]		suspend_t	*suspend_count*;
[10]		time_t	*sleep_time*;
[11]	};		
[12]	typedef struct thread_basic_info*		**thread_basic_info_t;**

DESCRIPTION

The **thread_basic_info** structure defines the basic information array for threads. The **thread_info** function returns this array for a specified thread.

FIELDS

user_time
: The total user run time for the thread.

system_time
: The total system run time for the thread.

cpu_usage
: Scaled CPU usage percentage for the thread.

policy
: Scheduling policy in effect

run_state
: The thread's run state. Possible values are:

TH_STATE_RUNNING
: The thread is running normally.

TH_STATE_STOPPED
: The thread is stopped.

TH_STATE_WAITING
: The thread is waiting normally.

TH_STATE_UNINTERRUPTIBLE
The thread is in an un-interruptible wait state.

TH_STATE_HALTED
The thread is halted at a clean point.

flags
Swap/idle flags for the thread. Possible values are:

TH_FLAGS_SWAPPED
The thread is swapped out.

TH_FLAGS_IDLE
The thread is an idle thread.

suspend_count
The current suspend count for the thread.

sleep_time
The number of seconds that the thread has been sleeping.

RELATED INFORMATION

Functions: **thread_info**.

Data structures: **policy_fifo_info**, **policy_rr_info**, **policy_timeshare_info**.

Status:

Believed to be stable

tvalspec

Structure — Defines format of system time values

SYNOPSIS

```
[1]  struct tvalspec
[2]  {
[3]          time_t          tv_sec;
[4]          nsec_t          tv_nsec;
[5]  };
[6]  typedef struct tvalspec  tvalspec_t;
```

DESCRIPTION

The **tvalspec** structure defines the format of the time structure supplied to or returned from the kernel. This definition conforms to the Posix 1003.4 *timespec* definition where the *tv_nsec* structure member is valid if ($0 \le tv\_nsec < 10^9$) and the time period described is (($tv\_sec \times 10^9$) + tv_nsec) nanoseconds.

FIELDS

tv_sec
: Seconds.

tv_nsec
: Nanoseconds.

RELATED INFORMATION

Functions: **clock_get_time, clock_set_time**, **clock_sleep**, **clock_alarm**, **clock_-alarm_reply.**

Data Structures: **mapped_tvalspec**.

vm_region_info

Status:

Believed to be stable

Structure — Defines the attributes of a task's memory region

SYNOPSIS

```
[1]  struct vm_region_info
[2]  {
[3]          vm_prot_t           protection;
[4]          vm_prot_t           max_protection;
[5]          vm_inherit_t        inheritance;
[6]          boolean_t           shared;
[7]          boolean_t           reserved;
[8]          vm_offset_t         offset;
[9]          vm_behavior_t       behavior;
[10] };
[11] typedef struct vm_region_info*    vm_region_info_t;
```

DESCRIPTION

The **vm_region_info** structure defines the attributes of a memory region returned by **vm_region**.

FIELDS

protection
The current protection for the region.

max_protection
The maximum protection allowed for the region.

inheritance
The inheritance attribute for the region.

shared
Shared indicator. If true, the region is shared by another task. If false, the region is not shared.

reserved
If true the region is protected from random allocation.

offset
The region's offset into the memory object. The region begins at this offset.

behavior
Expected reference pattern for the memory. Valid values are:

VM_BEHAVIOR_DEFAULT
The system's default behavior.

VM_BEHAVIOR_RANDOM
No particular order expected.

VM_BEHAVIOR_SEQUENTIAL
Forward sequential order.

VM_BEHAVIOR_RSEQNTL
Reverse sequential order.

RELATED INFORMATION

Functions: **vm_region, vm_inherit**, **vm_reserve**, **vm_behavior_set**, **vm_protect**.

vm_statistics

Status:

Believed to be stable

Structure — Defines statistics for the kernel's use of virtual memory

SYNOPSIS

```
struct vm_statistics
{
        vm_size_t                free_count;
        vm_size_t                active_count;
        vm_size_t                inactive_count;
        vm_size_t                wire_count;
        vm_size_t                zero_fill_count;
        vm_size_t                reactivations;
        vm_size_t                pageins;
        vm_size_t                pageouts;
        vm_size_t                faults;
        vm_size_t                cow_faults;
        vm_size_t                lookups;
        vm_size_t                hits;
};
typedef struct vm_statistics*    vm_statistics_t;
```

DESCRIPTION

The **vm_statistics** structure defines the statistics available on the kernel's use of virtual memory. The statistics record virtual memory usage since the kernel was booted.

For related information for a specific task, see the **task_basic_info** structure.

FIELDS

free_count
The total number of free pages in the system.

active_count
The total number of pages currently in use and pageable.

inactive_count
The number of inactive pages.

wire_count
The number of pages that are wired in memory and cannot be paged out.

zero_fill_count
The number of zero-fill pages.

reactivations
The number of reactivated pages.

pageins
The number of requests for pages from a pager (such as the i-node pager).

pageouts
The number of pages that have been paged out.

faults
The number of times the **vm_fault** routine has been called.

cow_faults
The number of copy-on-write faults.

lookups
The number of object cache lookups.

hits
The number of object cache hits.

NOTES

This structure is machine word length specific because of the memory sizes returned.

RELATED INFORMATION

Functions: **task_info**, **host_statistics**, **host_page_size**.

Data Structures: **task_basic_info**.

APPENDIX H

Co–Location Data Structures

This appendix discusses the specifics of the various structures used as a part of the kernel's co-location server interfaces.

Status:

Experimental: [OSF-RI MK6]

mig_stub_routine

Typedef — Defines the calling convention for the MIG-generated server stubs

SYNOPSIS

```
[1]  typedef void (*mig_stub_routine) (
[2]            mach_msg_header_t              *InHeadP;
[3]            mach_msg_header_t              *OutHeadP);


[4]  typedef mig_stub_routine_t               mig_impl_routine_t;
[5]  typedef kern_return_t                    (*mig_impl_routine_t)(void);
```

DESCRIPTION

The **mig_stub_routine** type is the definitionfor MIG-generated server stub routines. These routines: unpack the request message, call the server procedure, and pack the reply message.

FIELDS

InHeadP
Pointer to the header for the request message.

OutHeadP
Pointer to the header for the reply message.

RELATED INFORMATION

Functions: **mach_subsystem_create.**

Structures: **routine_descriptor**.

routine_arg_descriptor

Status:

Experimental: [OSF-RI MK6]

Structure — Defines the nature of each argument in the routine call

SYNOPSIS

```
typedef enum {
        ROUTINE_ARG_PORT,
        ROUTINE_ARG_FIXED_ARRAY,
        ROUTINE_ARG_VAR_ARRY
} routine_arg_type;
typedef enum {
        ROUTINE_ARG_IN,
        ROUTINE_ARG_OUT,
        ROUTINE_ARG_INOUT
} routine_arg_direction;
typedef unsigned int                    routine_arg_offset;


struct routine_arg_descriptor
{
        routine_arg_type                type;
        routine_arg_direction           direction;
        routine_arg_offset              offset;
} *routine_arg_descriptor_t;
```

DESCRIPTION

The **routine_arg_descriptor** structure is the definition for the handling of each argument passed to/from a routine.

FIELDS

type
: The type of argument (Port, Array, etc.).

direction
: Whether this is an inout, ouput, or inout argument.

offset
: Offset in the list of routine arguments.

RELATED INFORMATION

Functions: **mach_subsystem_create.**

Structures: **routine_descriptor, rpc_subsystem**.

Status:

Experimental:
[OSF-RI MK6]

routine_descriptor

Structure — Defines the nature of each subsystem routine

SYNOPSIS

```
[1]  structure routine_descriptor
[2]  {
[3]          mig_impl_routine_t              impl_routine;
[4]          mig_stub_routine_t              stub_routine;
[5]          unsigned int                    argc;
[6]          unsigned int                    descr_count;
[7]          struct routine_arg_descriptor   *arg_descr;
[8]          unsigned int                    max_reply_msg;
[9]  } *routine_descriptor_t;
```

DESCRIPTION

The **routine_descriptor** structure is the definition of each exported function within a subsystem. The structure contains information of the function location, message disassembly routine, and the nature of the argument signature.

FIELDS

impl_routine
: Server function pointer (code entry point).

stub_routine
: Unmarshalling function pointer.

argc
: Number of argument words.

descr_count
: Number of complex argument descriptors.

arg_descr
: Pointer to the beginning of the argument descriptor array (containing *descr_count* entries).

max_reply_msg
: The maximum size for the reply message.

RELATED INFORMATION

Functions: **mach_subsystem_create.**

Structures: **routine_arg_descriptor, rpc_subsystem**.

rpc_subsystem

Status:

Experimental: [OSF-RI MK6]

Structure — Defines the set of server routines in a subsystem

SYNOPSIS

```
struct rpc_subsystem
{
        struct subsystem                  *subsystem;
        mach_msg_id_t                     start;
        mach_msg_id_t                     end;
        unsigned int                      maxsize;
        vm_address_t                      base_addr;
        struct routine_descriptor         routine[1];
        struct routine_arg_descriptor     arg_descriptor[1];
} *rpc_subsystem_t;


#define RPC_SUBSYSTEM_NULL                ((rpc_subsystem_t) 0)
```

DESCRIPTION

The **rpc_subsystem** structure describes the set of server routines that can be invoked by mach_rpc() on the ports that are registered with the subsystem. For each routine, the routine number is given, along with the address of the implementation function in the server and a description of the arguments of the routine (i.e. it's signature).

This structure definition is only a template for what is really a variable-length structure (generated by MIG for each subsystem). The actual structures do not always have one entry in the *routine* array, and also have a varying number of entries in the arg_descr array. Each routine has an array of zero or more arg descriptors one for each complex arg. These arrays are all catenated together to form the *arg_descriptor* field of the subsystem struct. The arg_descr field of each routine entry points to a unique sub-sequence within this catenated array. The goal is to keep everything contiguous.

FIELDS

subsystem
: Reserved for system use.

start
: Minimum routine number.

end
: Maximum routine number (+1).

maxsize
: Maximum size of a message received from **mach_msg**.

base_addr
: The address of this **rpc_subsystem** structure in user space.

routine
: Starting point for the array of routin descriptors. The number of entires is equal to *start-end*.

arg_descriptor
: Starting point for the complex argument descriptors. The number of entires is equal to the sum of the descr_count fields for all of the routines.

RELATED INFORMATION

Functions: **mach_subsystem_create, mach_port_allocate_subsystem, mach_msg.**

Structures: **routine_arg_descriptor, routine_descriptor**.

APPENDIX I Error Return Values

This appendix lists the various kernel return values.

Error Code Format

An error code has the following format:

- system code (6 bits). The **err_get_system** (*err*) macro extracts this field.
- subsystem code (12 bits). The **err_get_sub** (*err*) macro extracts this field.
- error code (14 bits). The **err_get_code** (*err*) macro extracts this field.

The various system codes are:

- *err_kern* —kernel
- *err_us* — user space library
- *err_server*— user space servers
- *err_mach_ipc* — Mach-IPC errors
- *err_local* — user defined errors

A typical user error code definition would be:

#define SOMETHING_WRONG *err_local* | **err_sub** (13) | 1

MIG Stub Errors

MIG_ARRAY_TOO_LARGE

User specified array not large enough to hold returned array

MIG_BAD_ARGUMENTS

Message receiver found an invalid message size, invalid header fields or invalid descriptors. This could only happen if an invalidly formatted message (i.e., one that did not pass through Mach IPC) were passed to a MIG de-multiplexing routine.

MIG_BAD_ID

Bad message ID. This is only returned by MIG de-multiplexing routines when the message ID in the supplied message is not handled by that routine.

MIG_EXCEPTION

A server has raised an exception. [added from OSF/1 R1.3]

MIG_REMOTE_ERROR

A server has detected an error. [added from OSF/1 R1.3]

MIG_REPLY_MISMATCH

The message ID in a reply message is not 100 more than the message ID of the request message.

MIG_SERVER_DIED

Message recipient no longer exists, or the recipient destroyed the request message without replying.

MIG_TRAILER_ERROR

The message trailer has an unknown format. [added from OSF/1 R1.3]

MIG_TYPE_ERROR

The wrong number or size of data or rights was received.

Base IPC Status

MACH_MSG_SUCCESS

Normal IPC success. This is the same value as KERN_SUCCESS.

MACH_MSG_IPC_KERNEL

(mask bit) Kernel resource shortage handling an IPC capability.

MACH_MSG_IPC_SPACE

(mask bit) No room in IPC name space for another capability name.

MACH_MSG_VM_KERNEL

(mask bit) Kernel resource shortage handling out-of-line memory.

MACH_MSG_VM_SPACE

(mask bit) No room in VM address space for out-of-line memory.

IPC Send Errors

The following errors can occur when **mach_msg** is used with the MACH_SEND_MSG option. This is also the case for all function interfaces.

MACH_SEND_INTERRUPTED

Message send interrupted.

MACH_SEND_INVALID_DATA

Message buffer is unreadable.

MACH_SEND_INVALID_DEST

The destination port name in the message is MACH_PORT_NULL, MACH_-PORT_DEAD, names a null or dead right, names a port set or is a right whose type (receive, send or send-once) does not match the type specified.

MACH_SEND_INVALID_HEADER

A field in the message header had a bad value.

MACH_SEND_INVALID_MEMORY

An out-of-line memory region does not exist in the address space or is not readable.

MACH_SEND_INVALID_NOTIFY

The notify port name (MACH_SEND_CANCEL) specified to **mach_msg** is not a receive right.

MACH_SEND_INVALID_REPLY

The reply port name in the message is MACH_PORT_DEAD, names a null right, names a port set or is a right whose type (receive, send or send-once) does not match the type specified.

MACH_SEND_INVALID_RIGHT

A port name in the message body is MACH_PORT_DEAD, names a null right, names a port set or is a right whose type (receive, send or send-once) does not match the type specified.

MACH_SEND_INVALID_TRAILER

The trailer to be sent does not correspond to the current kernel format, or the sending task does not have the privilege to supply the message attributes.

MACH_SEND_INVALID_TYPE

An invalid msg-type specification was encountered. [added from OSF/1 R1.3]

MACH_SEND_MSG_TOO_SMALL

Message buffer doesn't contain a complete message.

MACH_SEND_NO_BUFFER

No kernel message buffer is available.

MACH_SEND_TIMED_OUT

Message not sent before time-out expired.

IPC Receive Errors

The following errors can be returned by **mach_msg** when used with the MACH_-RCV_MSG option. They can occur for kernel function interfaces.

MACH_RCV_BODY_ERROR

Error receiving kernel message body. See special bits.

MACH_RCV_HEADER_ERROR

Error receiving message header. See special bits.

MACH_RCV_IN_SET

The receive port name specified to **mach_msg** is a member of a port set.

MACH_RCV_INTERRUPTED

A software interrupt occurred.

MACH_RCV_INVALID_DATA

The message buffer was not writable.

MACH_RCV_INVALID_NAME

The receive port name specified to **mach_msg** is MACH_PORT_NULL, MACH_PORT_DEAD, names a null or dead right or is a right whose type (receive, send or send-once) does not match the type specified.

MACH_RCV_INVALID_NOTIFY

The notify port name (MACH_RCV_NOTIFY) specified to **mach_msg** is not a receive right.

MACH_RCV_INVALID_TRAILER

The trailer type desired, or the number of trailer elements desired, is not supported by the kernel.

MACH_RCV_INVALID_TYPE

One or more scatter list descriptors did not have the type matching the corresponding incoming message descriptor or had an invalid copy (disposition) field.

MACH_RCV_LIMITS

The combined size of all out-of-line regions, the total of all port rights or the total message size exceeds the transfer limit for the port.

MACH_RCV_OVERWRITE_ERROR

A region described by a receive overwrite descriptor does not exist or cannot be written.

MACH_RCV_PORT_CHANGED

Receive right specified to **mach_msg** was moved into a set during the receive.

MACH_RCV_PORT_DIED

Receive right (or set) specified to **mach_msg** was sent away/died during receive.

MACH_RCV_SCATTER_SMALL

One or more scatter list descriptors specified an overwrite region smaller than the corresponding incoming region.

MACH_RCV_TIMED_OUT

A message was not received within the time-out value.

MACH_RCV_TOO_LARGE

Message buffer is not large enough for the message.

Generic Kernel Errors

KERN_SUCCESS

Successful completion

KERN_INVALID_ARGUMENT

The function requested was not applicable to this type of object.

KERN_INVALID_CAPABILITY

The supplied right is dead, null or not of the proper type.

KERN_INVALID_VALUE

A parameter's value was out of range (or possibly ill-formed). Specific error return values are returned if the parameter's value is properly formed and in range, but not a usable value at this time.

KERN_RESOURCE_SHORTAGE

A system resource could not be allocated to fulfill this request. This failure may not be permanent.

Port Manipulation Errors

KERN_INVALID_NAME

The port name doesn't denote a right in the task.

KERN_INVALID_RIGHT

The port name denotes a right, but not an appropriate right.

KERN_INVALID_TASK

The target task is not an active task. [added from OSF/1 R1.3]

KERN_NAME_EXISTS

The port name already denotes a right in the task.

KERN_NO_ACCESS

A bogus access restriction was encountered. [added from OSF/1 R1.3]

KERN_NO_SPACE

The task's port name space is full.

KERN_NOT_DEPRESSED

A **thread_depress_abort** was requested on a thread which was not currebtly depressed. [added from OSF/1 R1.3]

KERN_NOT_IN_SET

The receive right is not a member of a port set.

KERN_NOT_RECEIVER

The task in question does not hold receive rights for the port argument. [added from OSF/1 R1.3]

KERN_RIGHT_EXISTS

The task already has send or receive rights for the port under another name.

KERN_UREFS_UNDERFLOW

The user-reference count would become negative.

KERN_UREFS_OVERFLOW

Operation would overflow limit on user-references.

Virtual Memory Manipulation Errors

KERN_INVALID_ADDRESS

Specified virtual address is not currently valid.

KERN_INVALID_OBJECT

The external memory manager failed to initialize the memory object. [added from OSF/1 R1.3]

KERN_MEMORY_ERROR

During a page fault, the memory object indicated that the data could not be returned. This failure may be temporary; future attempts to access this same data may succeed, as defined by the memory object.

KERN_MEMORY_FAILURE

During a page fault, the target address refers to a memory object that has been destroyed. This failure is permanent.

KERN_MEMORY_PRESENT

An external memory manager attempted to supply pages already cached by the kernel.

KERN_NO_SPACE

The task's address space is full (not sufficient free space) or the specified address range is already in use.

KERN_PROTECTION_FAILURE

Specified memory is valid, but does not permit the required forms of access or the protection being requested exceeds that permitted.

Random Kernel Errors

KERN_ABORTED

The operation was aborted.

KERN_ALREADY_WAITING

A thread is attempting to wait for an event for which there is already a waiting thread.

KERN_DEFAULT_SET

An attempt was made to destroy the default processor set.

KERN_EXCEPTION_PROTECTED

An attempt was made to fetch an exception port that is protected, or to abort a thread while processing a protected exception.

KERN_FAILURE

A catch-all error for implementation dependent failures.

KERN_INVALID_HOST

An argument supplied to assert system privilege was not a host control port.

KERN_INVALID_LEDGER

A ledger was required but not supplied.

KERN_INVALID_MEMORY_CONTROL

The port was not a memory cache control port.

KERN_INVALID_POLICY

The specified scheduling policy is not currently enabled for the processor set.

KERN_INVALID_PROCESSOR_SET

An argument supplied to assert processor set privilege was not a processor set control port.

KERN_INVALID_SECURITY

An argument supplied to assert security privilege was not a host security port.

KERN_POLICY_LIMIT

The specified scheduling attributes exceeds the thread's limits.

Kernel Device Errors

D_SUCCESS

Normal device return. This is the same value as KERN_SUCCESS.

D_ALREADY_OPEN

Exclusive-use device already open.

D_DEVICE_DOWN

Device has been shut down.

D_INVALID_OPERATION

No filter port was specified.

D_INVALID_RECNUM

Invalid record (block) number.

D_INVALID_SIZE

Invalid IO size.

D_IO_ERROR

Hardware IO error.

D_NO_MEMORY

A memory allocation related to an I/O operation has failed. [added from OSF/1 R1.3]

D_NO_SUCH_DEVICE

No device with that name, or the device is not operational.

D_OUT_OF_BAND

Out-of-band condition occurred on device (such as typing control-C).

D_READ_ONLY

Data cannot be written to this device.

D_WOULD_BLOCK

Operation would block, but D_NOWAIT set.

APPENDIX J

Object Index

APPENDIX K

Interface and Structure Index