APPENDIX A libmach Support Functions

This chapter describes support functions and macros found in **libmach.a** and **<mach.h>**.

current_task

current_task Macro — Returns the task self port

LIBRARY

libmach.a

#include <mach.h>

SYNOPSIS

mach_port_t **current_task**();

DESCRIPTION

The **current_task** macro returns send rights to the task's own port.

PARAMETERS

None

RETURN VALUE

Send rights to the task's port.

RELATED INFORMATION

Functions: **mach_task_self.**

mach_init

mach_init Function — Mach task related start-up.

LIBRARY

libmach.a

SYNOPSIS

int **mach_init**();

DESCRIPTION

The **mach_init** function performs MACH related task start-up. It also invokes MIG related start-up. This call is made by **_start** automatically when a task starts.

PARAMETERS

None

RETURN VALUE

Not meaningful.

RELATED INFORMATION

Functions: **_start**, **mig_init**.

mach_msg_destroy

mach_msg_destroy Function — Clean up data associated with a received message

LIBRARY

libmach.a

SYNOPSIS

void **mach_msg_destroy**(mach_msg_header_t **msg);*

DESCRIPTION

The **mach_msg_destroy** function de-allocates all port rights and out-of-line memory found in a received message. Send and send-once rights are de-allocated; receive rights have their reference count decremented.

PARAMETERS

msg
A received message.

RETURN VALUE

None.

RELATED INFORMATION

Functions: **mach_msg**.

mach_msg_server

mach_msg_server Function — A simple generic server message loop

LIBRARY

libmach.a

SYNOPSIS

mach_msg_return_t **mach_msg_server**(boolean_t (***demux*)
(mach_msg_header_t **request*,
mach_msg_header_t **reply*), mach_msg_size_t
max_size, mach_port_t *rcv_name*);

DESCRIPTION

The **mach_msg_server** function loops, reading messages from *rcv_name*, and passing them to the *demux* routine. The *demux* routine is called as follows:

(**demux*) (*request*, *reply*);

where:

request
: is a pointer to the message received from *rcv_name*.

reply
: is a pointer to an area (of size *max_size*) into which a reply message is to be placed.

The *demux* routine is declared to take **mach_msg_header_t** as arguments. It is actually passed **mig_reply_header_t** values which are cast accordingly.

A reply message will be sent only if the value for the *RetCode* field of the *reply* structure is KERN_SUCCESS and the value of the *msgh_remote_port* in the *reply* structure is other than MACH_PORT_NULL. An error in the message send or receive operation of other than MACH_SEND_INVALID_-DEST terminates the loop. The resultant error code is returned.

PARAMETERS

demux
: A pointer to a routine to be called for each message received.

max_size
: The maximum size message to receive.

rcv_name
: A receive right to a port.

RETURN VALUE

KERN_RESOURCE_SHORTAGE
Insufficient virtual address space for the receive and reply buffers.

Other MIG and **mach_msg** errors terminate the call.

RELATED INFORMATION

Functions: **mach_msg**.

Data structures: **mig_reply_header**.

mach_task_self

mach_task_self Macro — Returns the task self port

LIBRARY

libmach.a

#include <mach.h>

SYNOPSIS

mach_port_t **mach_task_self**();

DESCRIPTION

The **mach_task_self** macro returns send rights to the task's own port. The include file **<mach_init.h>** included by **<mach.h>** redefines the kernel function to simply return the value **mach_task_self_**, cached by the MACH runtime.

PARAMETERS

None

RETURN VALUE

Send rights to the task's port.

RELATED INFORMATION

Functions: **mach_task_self** (kernel call).

mig_dealloc_reply_port

mig_dealloc_reply_port Function — De-allocate the reply port for MIG interfaces

LIBRARY

libmach.a

SYNOPSIS

void **mig_dealloc_reply_port**();

DESCRIPTION

The **mig_dealloc_reply_port** function is called by MIG interfaces after a time-out on the reply port.

PARAMETERS

None

RETURN VALUE

None

RELATED INFORMATION

Functions: **mig_get_reply_port**.

mig_get_reply_port

mig_get_reply_port Function — Generate a reply port for MIG interfaces

LIBRARY

libmach.a

SYNOPSIS

mach_port_t **mig_get_reply_port**();

DESCRIPTION

The **mig_get_reply_port** function is called by MIG interfaces when they need a reply port.

PARAMETERS

None

RETURN VALUE

The MIG reply port

RELATED INFORMATION

Functions: **mig_dealloc_reply_port**.

mig_init

mig_init Function — Prepares the task to perform MIG related MACH IPC functions

LIBRARY

libmach.a

SYNOPSIS

void **mig_init**();

DESCRIPTION

The **mig_init** function prepares the task to use MIG related services. This call is made automatically via **_start** when the task begins.

PARAMETERS

None

RETURN VALUE

None.

RELATED INFORMATION

Functions: **_start**, **cthread_init**.

mig_strncpy

mig_strncpy Function — Copy a character string, null terminated, with count.

LIBRARY

libmach.a

SYNOPSIS

void **mig_strncpy** (char **dst*, char **src*, unsigned int *length*);

DESCRIPTION

The **mig_strncpy** function copies a character string from *src* to *dst*. The copy terminates either when *length*-1 characters have been copied, or when a null character is encountered, whichever comes first. This routine differs from **strncpy** in that the resulting string is always null terminated.

PARAMETERS

dst
: Destination of the copy.

src
: Source of the copy.

length
: Number of bytes to move.

RETURN VALUE

None.

RELATED INFORMATION

Functions: **strncpy**.

round_page

round_page Macro — Round a virtual address to a page boundary

LIBRARY

#include <mach.h> (**#include <mach_init.h>**)

SYNOPSIS

vm_offset_t **round_page** (vm_offset_t *x*);

DESCRIPTION

The **round_page** macro rounds its virtual address argument to the nearest page boundary.

PARAMETERS

x

Virtual address

RETURN VALUE

Rounded virtual address

RELATED INFORMATION

Functions: **trunc_page**.

trunc_page

trunc_page Macro — Truncate a virtual address to a page boundary

LIBRARY

#include <mach.h> (**#include <mach_init.h>**)

SYNOPSIS

vm_offset_t **trunc_page** (vm_offset_t *x*);

DESCRIPTION

The **trunc_page** macro truncates its virtual address argument down to the nearest page boundary.

PARAMETERS

x

Virtual address

RETURN VALUE

Truncated virtual address

RELATED INFORMATION

Functions: **round_page**.

vm_page_size

vm_page_size Global Variable — Contains the page size for the current task.

LIBRARY

libmach.a

SYNOPSIS

extern vm_size_t **vm_page_size;**

DESCRIPTION

The **vm_page_size** variable contains the task's page size, in bytes.

RELATED INFORMATION

Functions: **vm_statistics**.

APPENDIX B Data Structure Definitions

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.

host_basic_info

host_basic_info Structure — Defines basic information about a host

FILE

<mach.h> (**<mach/host_info.h>**)

SYNOPSIS

```
struct host_basic_info {
        int             max_cpus;
        int             avail_cpus;
        vm_size_t       memory_size;
        cpu_type_t      cpu_type;
        cpu_subtype_t   cpu_subtype;
};
```

typedef struct host_basic_info **host_basic_info_data_t**;

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

RELATED INFORMATION

Functions: **host_info.**

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

host_load_info

host_load_info Structure — Defines load information about a host

FILE

<mach.h> (**<mach/host_info.h>**)

SYNOPSIS

#define **CPU_STATE_USER** 0

#define **CPU_STATE_SYSTEM** 1

#define **CPU_STATE_IDLE** 2

```
struct host_load_info {
        long        avenrun[3];
        long        mach_factor[3];
};
```

typedef struct host_load_info **host_load_info_data_t**;

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_info.**

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

host_sched_info

host_sched_info Structure — Defines scheduling information about a host

FILE

<mach.h> (**<mach/host_info.h>**)

SYNOPSIS

```
struct host_sched_info {
        int         min_timeout;
        int         min_quantum;
};
```

typedef struct host_sched_info **host_sched_info_data_t**;

typedef struct host_sched_info ***host_sched_info_t**;

DESCRIPTION

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

FIELDS

min_timeout
: Minimum time-out, in milliseconds

min_quantum
: Minimum quantum, in milliseconds

RELATED INFORMATION

Functions: **host_info.**

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

mach_dead_name_notification

mach_dead_name_notification Structure — Defines the message received due to a dead name notification

FILE

<mach/notify.h>

SYNOPSIS

```
typedef     struct {
                    mach_msg_header_t not_header;
                    mach_msg_type_t not_type;
                    mach_port_t     not_port;
            } mach_dead_name_notification_t;
```

DESCRIPTION

The **mach_dead_name_notification** structure is returned (in a **mach_msg_receive** call) to notify that the named *not_port* is now a dead-name port. This notification must have been previously requested via **mach_port_request_notification**.

FIELDS

not_header
: Standard IPC receive header. The *msgh_kind* field will have the value MACH_MSGH_KIND_NOTIFICATION and the *msgh_id* field will be 72.

not_type
: The value MACH_MSG_TYPE_PORT_NAME.

not_port
: The port now dead.

RELATED INFORMATION

Functions: **mach_msg**, **mach_port_request_notification.**

Data Structures: **mach_msg_header**.

mach_msg_accepted_notification

mach_msg_accepted_notification Structure — Defines the message received due to a message-accepted message having been received

FILE

<mach/notify.h>

SYNOPSIS

```
typedef     struct {
                    mach_msg_header_t not_header;
                    mach_msg_type_t not_type;
                    mach_port_t     not_port;
            } mach_msg_accepted_notification_t;
```

DESCRIPTION

The **mach_msg_accepted_notification** structure is returned (in a **mach_msg_receive** call) to notify that the named *not_port* accepted the message previously forcibly queued for it via a message send operation specifying MACH_SEND_NOTIFY. If the destination port should be destroyed prior to receiving the message-accepted message, the notification would have been **mach_send_once_notification** instead.

FIELDS

not_header
: Standard IPC receive header. The *msgh_kind* field will have the value MACH_MSGH_KIND_NOTIFICATION and the *msgh_id* field will be 66.

not_type
: The value MACH_MSG_TYPE_PORT_NAME.

not_port
: The port for which the message-accepted request was accepted.

NOTES

(Note: This feature is currently planned for deletion.)

RELATED INFORMATION

Functions: **mach_msg.**

Data Structures: **mach_msg_header**.

mach_msg_header

mach_msg_header Structure — Defines the header portion for messages

FILE

<mach.h> (**<mach/message.h>**)

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_kind_t  msgh_kind;
                        mach_msg_id_t    msgh_id;
                } mach_msg_header_t;
```

DESCRIPTION

A Mach message consists of a fixed size message header, a **mach_msg_header_t**, followed by zero or more data items. Data items are typed. Each item has a type descriptor followed by the actual data (or an address of the data, for out-of-line memory regions).

There are two forms of type descriptors, a **mach_msg_type_t** and a **mach_msg_type_long_t**. The **mach_msg_type_long_t** type descriptor allows larger values for these fields. The *msgtl_header* field in the long descriptor is only used for its in-line, long-form, and de-allocate bits.

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 port rights or out-of-line memory regions. If it is not specified, then the message body carries no port rights or memory, no matter what the type descriptors may seem to indicate.

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

In the header of a received message, this field contains the message's size. The message size, a byte quantity, includes the message header, type descriptors, and in-line data. For out-of-line memory regions, the message size includes the size of the in-line address, not the size of the actual data region. There are no arbitrary limits on the size of a Mach message, the number of data items in a message, or the size of the data items.

msgh_remote_port

Specifies the destination port of the message. The field must carry a legitimate send or send-once right for a port.

msgh_local_port

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.

msgh_kind

Not set or read by the **mach_msg** call. The conventional meaning is to classify the message, using general categories and attributes. Notifications from the kernel carry the value MACH_MSGH_KIND_NOTIFICATION in this field.

msgh_id

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

NOTES

Simple messages are provided to handle in-line data. The sender copies the in-line data into the message structure, and the receiver usually copies it out.

Non-simple messages are provided to handle out-of-line data. Out-of-line data allows you to send port information or data blocks that are very large or of variable size. The kernel maps out-of-line data from the address space of the sender to the address space of the receiver. The kernel copies the data only if the sender or receiver subsequently modifies it. This is an example of copy-on-write data sharing.

RELATED INFORMATION

Functions: **mach_msg**, **mach_msg_receive**, **mach_msg_send**.

Data Structures: **mach_msg_type**, **mach_msg_type_long**.

mach_msg_type

mach_msg_type Structure — Defines the data descriptor for long data items in messages

FILE

<mach.h> (**<mach/message.h>**)

SYNOPSIS

```
typedef    struct {
                     unsigned int    msgt_name: 8,
                                     msgt_size: 8,
                                     msgt_number: 12,
                                     msgt_inline: 1,
                                     msgt_longform: 1,
                                     msgt_deallocate: 1,
                                     msgt_unused: 1;
           } mach_msg_type_t;
```

DESCRIPTION

Each data item in a MACH IPC message has a type descriptor, a **mach_msg_type_t** or a **mach_msg_type_long_t**. The **mach_msg_type_long_t** type descriptor allows larger values for these fields.

FIELDS

msgt_name
Specifies the data's type. The following types are predefined:

MACH_MSG_TYPE_UNSTRUCTURED
un-interpreted data (32 bits)

MACH_MSG_TYPE_BIT
single bit

MACH_MSG_TYPE_BOOLEAN
boolean value (32 bits)

MACH_MSG_TYPE_INTEGER_16
16 bit integer

MACH_MSG_TYPE_INTEGER_32
32 bit integer

MACH_MSG_TYPE_CHAR
single character

MACH_MSG_TYPE_BYTE
: 8-bit byte

MACH_MSG_TYPE_INTEGER_8
: 8-bit integer

MACH_MSG_TYPE_REAL
: floating value (32 bits)

MACH_MSG_TYPE_STRING
: null terminated

MACH_MSG_TYPE_STRING_C
: null terminated

MACH_MSG_TYPE_PORT_NAME
: type of **mach_port_t**. This is the type of the name for a port, not the type to specify if a port right is to be specified.

MACH_MSG_TYPE_MOVE_RECEIVE
: move the name receive right

MACH_MSG_TYPE_MOVE_SEND
: move the named send right

MACH_MSG_TYPE_MOVE_SEND_ONCE
: move the named send-once right

MACH_MSG_TYPE_COPY_SEND
: make a copy of the named send right

MACH_MSG_TYPE_MAKE_SEND
: make a send right from the named receive right

MACH_MSG_TYPE_MAKE_SEND_ONCE
: make a send-once right from the named send or receive right

The last six types specify port rights, and receive special treatment. The type MACH_MSG_TYPE_PORT_NAME describes port right names, when no rights are being transferred, but just names. For this purpose, it should be used in preference to MACH_MSG_TYPE_INTEGER_32.

msgt_size
: Specifies the size of each datum, in bits. For example, the *msgt_size* of MACH_MSG_TYPE_INTEGER_32 data is 32.

msgt_number
: Specifies how many data elements comprise the data item. Zero is a legitimate number. The total length specified by a type descriptor is

(*msgt_size* * *msgt_number*), rounded up to an integral number of bytes. In-line data is then padded to an integral number of long-words. This ensures that type descriptors always start on long-word boundaries. It implies that message sizes are always an integral multiple of a long-word's size.

msgt_inline
: When FALSE, specifies that the data actually resides in an out-of-line region. The address of the data region follows the type descriptor in the message body. The *msgt_name*, *msgt_size*, and *msgt_number* fields describe the data region, not the address.

msgt_longform
: Specifies, when TRUE, that this type descriptor is a **mach_msg_type_long_t** instead of a **mach_msg_type_t.**

msgt_deallocate
: Used with out-of-line regions. When TRUE, it specifies the data region should be de-allocated from the sender's address space (as if with **vm_deallocate**) when the message is sent.

msgt_unused
: Not used, should be zero.

RELATED INFORMATION

Functions: **mach_msg**, **mach_msg_receive**, **mach_msg_send**.

Data Structures: **mach_msg_header**, **mach_msg_type_long**.

mach_msg_type_long

mach_msg_type_long Structure — Defines the data descriptor for long data items in messages

FILE

<mach.h> (**<mach/message.h>**)

SYNOPSIS

```
typedef        struct        {
                                 mach_msg_type_t msgtl_header;
                                 unsigned short  msgtl_name;
                                 unsigned short  msgtl_size;
                                 unsigned int    msgtl_number;
               } mach_msg_type_long_t;
```

DESCRIPTION

Each data item has a type descriptor, a **mach_msg_type_t** or a **mach_msg_type_long_t**. The **mach_msg_type_long_t** type descriptor allows larger values for these fields. The *msgtl_header* field in the long descriptor is only used for its in-line, long-form, and de-allocate bits.

FIELDS

msgtl_header
: A header in common with **mach_msg_type_t**. When the *msgt_longform* bit in the header is TRUE, this type descriptor is a **mach_msg_type_long_t** instead of a **mach_msg_type_t**. The *msgt_name*, *msgt_size*, and *msgt_number* fields should be zero. Instead, **mach_msg** uses the following: *msgtl_name*, *msgtl_size*, and *msgtl_number* fields.

msgtl_name
: Specifies the data's type. The defined values are the same as those for **mach_msg_type**.

msgtl_size
: Specifies the size of each datum, in bits. For example, the *msgtl_size* of MACH_MSG_TYPE_INTEGER_32 data is 32.

msgtl_number
: Specifies how many data elements comprise the data item. Zero is a legitimate number. The total length specified by a type descriptor is (*msgtl_size* * *msgtl_number*), rounded up to an integral number of bytes. In-line data is then padded to an integral number of longwords. This ensures that type descriptors always start on long-word boundaries. It implies that message sizes are always an integral multiple of a long-word's size.

RELATED INFORMATION

Functions: **mach_msg**, **mach_msg_receive**, **mach_msg_send**.

Data Structures: **mach_msg_header**, **mach_msg_type**.

mach_no_senders_notification

mach_no_senders_notification Structure — Defines the message received due to a receive port having no more known senders

FILE

<mach/notify.h>

SYNOPSIS

```
typedef         struct {
                        mach_msg_header_t not_header;
                        mach_msg_type_t not_type;
                        unsigned int not_count;
                } mach_no_senders_notification_t;
```

DESCRIPTION

The **mach_no_senders_notification** structure is returned (in a **mach_msg_receive** call) to notify that a port has lost its last sender. The port that has lost its senders is not included in the message, but its make-send count is. This notification must have been previously requested via **mach_port_request_notification**.

FIELDS

not_header
: Standard IPC receive header. The *msgh_kind* field will have the value MACH_MSGH_KIND_NOTIFICATION and the *msgh_id* field will be 70.

not_type
: The value MACH_MSG_TYPE_INTEGER_32.

not_count
: The value the port's make-send count had when it was generated.

RELATED INFORMATION

Functions: **mach_msg**, **mach_port_request_notification.**

Data Structures: **mach_msg_header**.

mach_port_destroyed_notification

mach_port_destroyed_notification Structure — Defines the message received due to a port being destroyed

FILE

<mach/notify.h>

SYNOPSIS

```
typedef     struct {
                        mach_msg_header_t not_header;
                        mach_msg_type_t not_type;
                        mach_port_t     not_port;
            } mach_port_destroyed_notification_t;
```

DESCRIPTION

The **mach_port_destroyed_notification** structure is returned (in a **mach_msg_receive** call) to notify that the named *not_port* receive right would have been destroyed. Instead of being destroyed, though, the receive right is send in a port destroyed notification, if this notification had been previously requested by **mach_port_request_notification**.

FIELDS

not_header
: Standard IPC receive header. The *msgh_kind* field will have the value MACH_MSGH_KIND_NOTIFICATION and the *msgh_id* field will be 69.

not_type
: The value MACH_MSG_TYPE_PORT_RECEIVED.

not_port
: The port that would have been destroyed.

NOTES

(Note: This feature is currently planned for deletion.)

RELATED INFORMATION

Functions: **mach_msg**, **mach_port_request_notification.**

Data Structures: **mach_msg_header**.

mach_port_status

mach_port_status Structure — Defines information for a port

FILE

<mach.h> (**<mach/port.h>**)

SYNOPSIS

```
struct mach_port_status {
        mach_port_t            mps_pset;
        mach_port_mscount_t    mps_mscount;
        mach_port_msgcount_t   mps_qlimit;
        mach_port_msgcount_t   mps_msgcount;
        mach_port_rights_t     mps_sorights;
        boolean_t              mps_srights;
        boolean_t              mps_pdrequest;
        boolean_t              mps_nsrequest;
};



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_mscount
: Make-send count

mps_qlimit
: Queue limit

mps_msgcount
: Number in the queue

mps_sorights
: How many send-once rights

mps_srights
: True if send rights exist

mps_pdrequest
: True if there is a port-deleted requested

mps_nsrequest
: True if no-senders requested

RELATED INFORMATION

Functions: **mach_port_get_receive_status.**

mach_send_once_notification

mach_send_once_notification Structure — Defines the message received when a send-once right is destroyed or de-allocated

FILE

<mach/notify.h>

SYNOPSIS

```
typedef     struct {
                        mach_msg_header_t not_header;
            } mach_send_once_notification_t;
```

DESCRIPTION

The **mach_send_once_notification** structure is returned (in a **mach_msg_receive** call) to notify that a send-once right was destroyed or de-allocated prior to the message being delivered. This notification is received only if the MACH_SEND_NOTIFY option was specified to **mach_msg**.

FIELDS

not_header
: Standard IPC receive header. The *msgh_kind* field will have the value MACH_MSGH_KIND_NOTIFICATION and the *msgh_id* field will be 71.

RELATED INFORMATION

Functions: **mach_msg.**

Data Structures: **mach_msg_header**.

mig_reply_header

mig_reply_header Structure — Defines the true type of information passed in and out of **mach_msg_server**

FILE

<mach/mig_errors.h>

SYNOPSIS

typedef struct {
mach_msg_header_t *Head*;
mach_msg_type_t *RetCodeType*;
kern_return_t *RetCode*;
} **mig_reply_header_t**;

DESCRIPTION

The **mig_reply_header** structure defines the format of the data interface between **mach_msg_server** and the various MIG generated servers it calls.

FIELDS

Head
The actual MACH IPC message

RetCodeType
Not used

RetCode
A return code to **mach_msg_server**, indicating the disposition of the return message.

RELATED INFORMATION

Functions: **mach_msg_server.**

Data structures: **mach_msg_header.**

processor_basic_info

processor_basic_info Structure — Defines the basic information about a processor.

FILE

<mach.h> (**<mach/processor_info.h>**)

SYNOPSIS

```
struct processor_basic_info {
        cpu_type_t       cpu_type;
        cpu_subtype_t    cpu_subtype;
        boolean_t        running;
        int              slot_num;
        boolean_t        is_master;
};
```

```
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

processor_set_basic_info Structure — Defines the basic information about a processor set.

FILE

<mach.h> (**<mach/processor_info.h>**)

SYNOPSIS

```
struct processor_set_basic_info {
        int        processor_count;
        int        task_count;
        int        thread_count;
        int        load_average;
        int        mach_factor;
};
```

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

task_count
: Number of tasks currently assigned to this processor set

thread_count
: Number of threads currently assigned to this processor set

load_average
: Scaled

mach_factor
: Scaled

RELATED INFORMATION

Functions: **processor_set_info**.

Data Structures: **processor_set_sched_info**.

processor_set_sched_info

processor_set_sched_info Structure — Defines the scheduling information about a processor set.

FILE

<mach.h> (**<mach/processor_info.h>**)

SYNOPSIS

```
struct processor_set_sched_info {
        int         policies;
        int         max_priority;
};
```

```
typedef struct processor_set_sched_info *processor_set_sched_info_t;
```

DESCRIPTION

The **processor_set_sched_info** structure defines the global scheduling information available about a processor set.

FIELDS

policies
: Allowed policies

max_priority
: Maximum scheduling priority for new threads

RELATED INFORMATION

Functions: **processor_set_info**.

Data Structures: **processor_set_basic_info**.

task_basic_info

task_basic_info Structure — Defines basic information for tasks

FILE

<mach.h> (**<mach/task_info.h>**)

SYNOPSIS

```
struct task_basic_info {
        int suspend_count;
        int base_priority;
        vm_size_t virtual_size;
        vm_size_t resident_size;
        time_value_t user_time;
        time_value_t system_time;
};



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.

base_priority
: The base scheduling priority 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.

RELATED INFORMATION

Functions: **task_info**.

Data Structures: **task_thread_times_info**.

task_thread_times_info

task_thread_times_info Structure — Defines thread execution times information for tasks

FILE

<mach.h> (**<mach/task_info.h>**)

SYNOPSIS

struct **task_thread_times_info** {
        time_value_t *user_time*;
        time_value_t *system_time*;
};

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**.

thread_basic_info

thread_basic_info Structure — Defines basic information for threads

FILE

<mach.h> (**<mach/thread_info.h>**)

SYNOPSIS

struct **thread_basic_info** {
time_value_t *user_time*;
time_value_t *system_time*;
int *cpu_usage*;
int *base_priority*;
int *cur_priority*;
int *run_state*;
int *flags*;
int *suspend_count*;
long *sleep_time*;
};

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.

base_priority
The base scheduling priority for the thread.

cur_priority
The current scheduling priority for the thread.

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: **thread_sched_info**.

thread_sched_info

thread_sched_info Structure — Defines scheduling information for threads

FILE

<mach.h> (**<mach/thread_info.h>**)

SYNOPSIS

```
struct thread_sched_info {
            int policy;
            int data;
            int base_priority;
            int max_priority;
            int cur_priority;
            boolean_t depressed;
            int depress_priority;
};



typedef struct thread_sched_info*thread_sched_info_t;
```

DESCRIPTION

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

FIELDS

policy
: Scheduling policy in effect

data
: Associated data for the scheduling policy

base_priority
: Base scheduling priority

max_priority
: Maximum 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**.

Data Structures: **thread_basic_info**.

vm_statistics

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

FILE

<mach.h> (**<mach/vm_statistics.h>**)

SYNOPSIS

```
struct vm_statistics {
        long pagesize;
        long free_count;
        long active_count;
        long inactive_count;
        long wire_count;
        long zero_fill_count;
        long reactivations;
        long pageins;
        long pageouts;
        long faults;
        long cow_faults;
        long lookups;
        long hits;
};
```

```
typedef struct vm_statistics vm_statistics_data_t;

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.

You can also find *pagesize* by using the global variable *vm_page_size*. This variable is set at task initialization and remains constant for the life of the task.

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

FIELDS

pagesize
: The virtual page size, in bytes.

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.

RELATED INFORMATION

Functions: **task_info**, **vm_statistics**.

Data Structures: **task_basic_info**.

APPENDIX C Debugging Interface Specifications

This appendix describes interfaces of a debugging nature. These interfaces are often machine dependent, highly subject to change, often require various kernel options to have been configured, and may very well disappear altogether in the future.

host_ipc_hash_info

host_ipc_hash_info Function — Returns information about the global reverse hash table

LIBRARY

Generated by **mach_debug.defs**.

SYNOPSIS

kern_return_t **host_ipc_hash_info** (host_t *host,* ipc_info_bucket_array_t **info,* mach_msg_type_number_t **infoCnt)*;

DESCRIPTION

The **host_ipc_hash_info** function returns information describing the use of the kernel's IPC reverse hash table.

See **ipc_info_bucket** for a description of the structure used.

PARAMETERS

host
: The name port for the host for which the IPC information is returned.

info
: [out array] An array of IPC bucket information.

infoCnt
: [out] The size of the *info* array.

NOTES

This interface is present only if the kernel was built with the MACH_IPC_DEBUG option.

info 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 statistics have been returned.

KERN_INVALID_HOST
: The *host* is null.

KERN_RESOURCE_SHORTAGE
: The kernel could not allocate sufficient memory.

RELATED INFORMATION

Data structures: **ipc_info_bucket**.

host_ipc_marequest_info

host_ipc_marequest_info Function — Returns information about the message accepted request hash table.

LIBRARY

Generated by **mach_debug.defs**.

SYNOPSIS

kern_return_t **host_ipc_marequest_info**(host_t *host,* unsigned int *max_requests*, ipc_info_bucket_array_t **info,* mach_msg_type_number_t **infoCnt)*;

DESCRIPTION

The **host_ipc_marequest_info** function returns information describing the use of the kernel's message accepted request hash table. A message accepted request is one that is created when a sender is forcing a message with MACH_SEND_NOTIFY.

See **ipc_info_bucket** for a description of the structure used.

PARAMETERS

host
: The name port for the host for which zone information is returned.

max_requests
: [out] The maximum number of message accepted requests allowed

info
: [out array] An array describing the kernel's use of the message accepted request hash table

infoCnt
: [out] The size of the *info* array.

NOTES

This interface is present only if the kernel was built with the MACH_IPC_DEBUG option.

info 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 statistics have been returned.

KERN_INVALID_HOST
The *host* is null.

KERN_RESOURCE_SHORTAGE
The kernel could not allocate sufficient memory.

RELATED INFORMATION

Data structures: **ipc_info_bucket**.

host_stack_usage

host_stack_usage Function — Returns kernel stack usage statistics

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **host_stack_usage** (host_t *host,* vm_size_t **reserved*, unsigned int **total,* vm_size_t **space*, vm_size_t **residentp*, vm_size_t **maxusagep,* vm_size_t **maxstackp)*;

DESCRIPTION

The **host_stack_usage** function returns kernel stack statistics for a host.

PARAMETERS

host
: A host name port.

reserved
: [out] Number of reserved stacks.

total
: [out] Number of kernel stacks.

space
: [out] Total number of kernel stack pages.

residentp
: [out] Number of memory resident stack pages

maxusagep
: [out] Maximum number of pages used in any one kernel stack.

maxstackp
: [out] The kernel stack using the most pages.

RETURN VALUE

KERN_SUCCESS
: The statistics have been returned.

KERN_RESOURCE_SHORTAGE
: The kernel could not allocate sufficient memory.

KERN_INVALID_HOST
: The host is invalid.

host_zone_info

host_zone_info Function — Returns memory allocation zone information.

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **host_zone_info** (host_t *host,* zone_name_array_t
**zone_names,* mach_msg_type_number_t
**zone_namesCnt,* zone_info_array_t **zone_info,*
mach_msg_type_number_t **zone_infoCnt)*;

DESCRIPTION

The **host_zone_info** function returns information describing the use of the kernel's memory zones. Memory zones are areas of memory divided into fixed sized blocks that are used for fast memory allocation and de-allocation within the kernel.

PARAMETERS

host
: The name port for the host for which zone information is desired.

zone_names
: [in/out array] An array naming the kernel's memory zones.

zone_namesCnt
: [in/out] The size of the *zone_names* array.

zone_info
: [in/out array] An array describing the kernel's memory zones.

zone_infoCnt
: [in/out] The size of the *zone_info* array.

RETURN VALUE

KERN_SUCCESS
: The statistics have been returned.

KERN_INVALID_HOST
: The *host* is null.

KERN_RESOURCE_SHORTAGE
: The kernel could not allocate sufficient memory.

RELATED INFORMATION

Data structures: **zone_info**, **zone_name**.

mach_port_dnrequest_info

mach_port_dnrequest_info Function — Returns information about the dead-name rights associated with a receive right

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **mach_port_dnrequest_info**(ipc_space_t *task,*
mach_port_name_t *name,* unsigned int **total,*
unsigned int **used)*;

DESCRIPTION

The **mach_port_dnrequest_info** function returns information about the dead-name rights associated with the named receive right.

PARAMETERS

task
: The task for which information is requested.

name
: The name of the receive port to query.

total
: [out] Total size of dead-name table

used
: [out] Amount of table used

NOTES

This interface is present only if the kernel was built with the MACH_IPC_DEBUG option.

RETURN VALUE

KERN_SUCCESS
: The information has been returned.

KERN_INVALID_TASK
: The specified task is invalid.

KERN_INVALID_NAME
: The port name does not exist.

KERN_INVALID_RIGHT
: The name doesn't denote a receive right

mach_port_get_srights

mach_port_get_srights Function — Returns the number of extant send rights

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **mach_port_get_srights**(ipc_space_t *task,* mach_port_name_t *name,* mach_port_rights_t **srights)*;

DESCRIPTION

The **mach_port_get_srights** function returns the number of extant send rights associated with the named receive right.

PARAMETERS

task
: The task for which information is requested.

name
: The name of the receive port to query.

srights
: [out] Amount of table used

NOTES

This interface is present only if the kernel was built with the MACH_IPC_DEBUG option.

RETURN VALUE

KERN_SUCCESS
: The information has been returned.

KERN_INVALID_TASK
: The specified task is invalid.

KERN_INVALID_NAME
: The port name does not exist.

KERN_INVALID_RIGHT
: The name doesn't denote a receive right

RELATED INFORMATION

Functions: **mach_port_get_receive_status**, **mach_port_get_refs**.

mach_port_space_info

mach_port_space_info Function — Returns information about a port space

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **mach_port_space_info**(ipc_space_t *task,* ipc_info_space_t **info,* ipc_info_name_array_t **table_info,* mach_msg_type_number_t **table_infoCnt,* ipc_info_tree_name_array_t **tree_info,* mach_msg_type_number_t **tree_infoCnt)*;

DESCRIPTION

The **mach_port_space_info** function returns information about a port space.

PARAMETERS

task
: The task for which information is requested.

info
: [out] Returned information about the IPC space.

table_info
: [out array] A returned array of information about each port name.

table_infoCnt
: [out array] The size of the *table_info* array

tree_info
: [out array] A returned array of information about each tree element in the IPC port space.

tree_infoCnt
: [out] The size of the *tree_info* array

NOTES

This interface is present only if the kernel was built with the MACH_IPC_DEBUG option.

RETURN VALUE

KERN_SUCCESS
: The information has been returned.

KERN_INVALID_TASK
: The specified task is invalid.

KERN_RESOURCE_SHORTAGE
The kernel couldn't allocate sufficient memory.

RELATED INFORMATION

Data structures: **ipc_info_name**, **ipc_info_space**, **ipc_info_tree_name**.

mach_vm_region_info

mach_vm_region_info Function — Returns information on a region of virtual memory, including the chain of objects rooted at that region

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **mach_vm_region_info**(vm_task_t *target_task,* vm_address_t *address,* vm_info_region_t **region,* vm_info_object_array_t **objects,* mach_msg_type_number_t **objectsCnt)*;

DESCRIPTION

The **mach_vm_region_info** function returns information on a region within the specified task's address space. It also includes information on the various memory objects rooted at this region.

The function begins looking at *address* and continues until it finds an allocated region. If the input address is within a region, the function uses the start of that region.

PARAMETERS

target_task
: The task whose address space contains the region.

address
: The address at which to start looking for a region.

region
: [out] Information on the region found.

objects
: [out array] An array of information about the objects rooted at this region.

objectsCnt
: [out] The number of elements in the *objects* array.

NOTES

This interface is present only if the kernel was built with the MACH_VM_DEBUG option.

region and *objects* are 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
: A region has been located and its information returned.

KERN_NO_SPACE
: There is no region at or beyond the specified starting address.

KERN_INVALID_TASK
: The map is null.

KERN_RESOURCE_SHORTAGE
: The kernel cannot allocate sufficient memory to extract the information.

RELATED INFORMATION

Functions: **vm_allocate**, **vm_deallocate**, **vm_inherit**, **vm_protect**.

Data Structures: **vm_info_region, vm_info_object**.

processor_set_stack_usage

processor_set_stack_usage Function — Returns kernel stack usage statistics

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **processor_set_stack_usage** (processor_set_t *pset,* unsigned int **total,* vm_size_t **space*, vm_size_t **residentp*, vm_size_t **maxusagep,* vm_size_t **maxstackp)*;

DESCRIPTION

The **processor_set_stack_usage** function returns kernel stack statistics for a processor set.

PARAMETERS

pset
: A processor set name port.

total
: [out] Number of kernel stacks.

space
: [out] Total number of kernel stack pages.

residentp
: [out] Number of memory resident stack pages

maxusagep
: [out] Maximum number of pages used in any one kernel stack.

maxstackp
: [out] The kernel stack using the most pages.

RETURN VALUE

KERN_SUCCESS
: The statistics have been returned.

KERN_RESOURCE_SHORTAGE
: The kernel could not allocate sufficient memory.

KERN_INVALID_ARGUMENT
: The processor set does not exist or is not active.

vm_mapped_pages_info

vm_mapped_pages_info Function — Returns a list of resident pages for a task

LIBRARY

Generated by **mach_debug.defs.**

SYNOPSIS

kern_return_t **vm_mapped_pages_info**(task_t *target_task,*
page_address_array_t *v*m_pages,*
mach_msg_type_number_t **vm_pagesCnt)*;

DESCRIPTION

The **vm_mapped_pages_info** function returns a list of the virtual addresses of pages in the task that are resident in physical memory.

PARAMETERS

target_task
A task port for which the resident set is to be returned.

vm_pages
[out array] The returned array of page mappings. Each element is the virtual address of a resident page in the task. The elements are in no particular order.

vm_pagesCnt
[out] The size of *vm_pagesCnt*.

NOTES

This interface is present only if the kernel was built with the MACH_VM_DEBUG option.

This call returns meaningful information currently only on a VAX.

vm_pages 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 mappings have been returned.

KERN_RESOURCE_SHORTAGE
The kernel could not allocate sufficient memory.

APPENDIX D

Debugging Data Structure Definitions

This appendix discusses the specifics of the various structures used as a part of the kernel's various debugging interfaces.

ipc_info_bucket

ipc_info_bucket Structure — Defines IPC hash bucket statistics

FILE

<mach_debug/ipc_info.h>

SYNOPSIS

```
struct ipc_info_bucket{
        unsigned int    iib_count;
};

typedef struct ipc_info_bucket ipc_info_bucket_t;

typedef ipc_info_bucket_t *ipc_info_bucket_array_t;
```

DESCRIPTION

The **ipc_info_bucket** structure defines the statistics available on IPC global reverse hash table bucket usage.

FIELDS

iib_count
: Number of records in bucket

RELATED INFORMATION

Functions: **host_ipc_hash_info**, **host_ipc_marequest_info**.

ipc_info_name

ipc_info_name Structure — Defines IPC information about a port name

FILE

<mach_debug/ipc_info.h>

SYNOPSIS

struct **ipc_info_name** {
mach_port_t *iin_name*;
boolean_t *iin_collision*;
boolean_t *iin_compat*;
boolean_t *iin_marequest*;
mach_port_type_t *iin_type*;
mach_port_urefs_t *iin_urefs*;
vm_offset_t *iin_object*;
unsigned int *iin_next*;
unsigned int *iin_hash*;
};

typedef struct ipc_info_name **ipc_info_name_t**;

typedef ipc_info_name_t ***ipc_info_name_array_t**;

DESCRIPTION

The **ipc_info_name** structure defines information available about a port name.

FIELDS

iin_name
Port name, including generation number

iin_collision
True if there is a collision at this entry

iin_compat
True if this is a compatibility-mode entry

iin_marequest
True if there is an extant msg-accepted request

iin_type
Straight port type

iin_urefs
User-references

iin_object
: Object address

iin_next
: Message accepted request/next in free list

iin_hash
: Hash index

RELATED INFORMATION

Functions: **mach_port_space_info**.

ipc_info_space

ipc_info_space Structure — Defines information about a port space

FILE

<mach_debug/ipc_info.h>

SYNOPSIS

struct **ipc_info_space** {
unsigned int *iis_genno_mask*;
unsigned int *iis_table_size*;
unsigned int *iis_table_next*;
unsigned int *iis_tree_size*;
unsigned int *iis_tree_small*;
unsigned int *iis_tree_hash*;
};

typedef struct ipc_info_space **ipc_info_space_t**;

DESCRIPTION

The **ipc_info_space** structure defines information describing a port space.

FIELDS

iis_genno_mask
Generation number mask

iis_table_size
Size of table

iis_table_next
Next possible size of table

iis_tree_size
Size of tree

iis_tree_small
Number of small entries in tree

iis_tree_hash
Number of hashed entries in tree

RELATED INFORMATION

Functions: **mach_port_space_info**.

ipc_info_tree_name

ipc_info_tree_name Structure — Defines information on a port space tree element

FILE

<mach_debug/ipc_info.h>

SYNOPSIS

struct **ipc_info_tree_name** {
  ipc_info_name_t *iitn_name*;
  mach_port_t *iitn_lchild*;
  mach_port_t *iitn_rchild*;
};

typedef struct ipc_info_tree_name **ipc_info_tree_name_t**;

typedef ipc_info_tree_name_t ***ipc_info_tree_name_array_t**;

DESCRIPTION

The **ipc_info_tree_name** structure defines information about a tree element in an IPC port space.

FIELDS

iitn_name
: Port name, including generation number

iitn_lchild
: Name of left child

iitn_rchild
: Name of right child

RELATED INFORMATION

Functions: **mach_port_space_info.**

vm_info_object

vm_info_object Structure — Defines information pertaining to a memory object

FILE

<mach_debug/vm_info.h>

SYNOPSIS

struct **vm_info_object** {
vm_offset_t *vio_object*;
vm_size_t *vio_size*;
unsigned int *vio_ref_count*;
unsigned int *vio_resident_page_count*;
unsigned int *vio_absent_count*;
vm_offset_t *vio_copy*;
vm_offset_t *vio_shadow*;
vm_offset_t *vio_shadow_offset*;
vm_offset_t *vio_paging_offset*;
memory_object_copy_strategy_t *vio_copy_strategy*;
vm_offset_t *vio_last_alloc*;
unsigned int *vio_paging_in_progress*;
boolean_t *vio_pager_created*;
boolean_t *vio_pager_initialized*;
boolean_t *vio_pager_ready*;
boolean_t *vio_can_persist*;
boolean_t *vio_internal*;
boolean_t *vio_temporary*;
boolean_t *vio_alive*;
boolean_t *vio_lock_in_progress*;
boolean_t *vio_lock_restart*;
boolean_t *vio_single_use*;
};

typedef struct vm_info_object **vm_info_object_t**;

typedef vm_info_object_t ***vm_info_object_array_t**;

DESCRIPTION

The **vm_info_object** structure defines the information available on a memory object.

FIELDS

vio_object
Address of the kernel's *vm_object* structure describing this object. This structure cannot be directly referenced by user code. This field is supplied as a means of distinguishing between memory objects.

vio_size
Size of the memory object (valid if internal)

vio_ref_count
Number of references

vio_resident_page_count
Number of memory resident pages

vio_absent_count
Number of pages requested but not filled. This includes pages upon which page faults occurred, as well as pages for which page faults have not yet occurred, but for which data is being supplied via **memory_object_data_provided**

vio_copy
True if this memory object is a copy object

vio_shadow
True if this memory object is a shadow object

vio_shadow_offset
Offset into shadow object

vio_paging_offset
Offset into memory object

vio_copy_strategy
How the kernel should handle copying of regions associated with the memory object. Possible 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_CALL
Notify the memory manager (via **memory_object_copy**) before copying any 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.

vio_last_alloc
Offset of last page allocation

vio_paging_in_progress
Count of busy pages in object

vio_pager_created
True if a pager port has been established for the object

vio_pager_initialized
True when this object is completely mapped into an address space

vio_pager_ready
True when the pager for this object asserts its readiness

vio_can_persist
True if the object can persist after all tasks have un-mapped it

vio_internal
True if this memory object depends on the default memory manager

vio_temporary
True if the memory object is temporary (managed by the default memory manager)

vio_alive
False if the object is being terminated

vio_lock_in_progress
True if a multi-page lock request is in progress

vio_lock_restart
True if a lock request in progress should restart its search

vio_single_use
True if only one address map has access to this object (not used yet)

RELATED INFORMATION

Functions: **mach_vm_region_info**.

Data Structures: **vm_info_region**.

vm_info_region

vm_info_region Structure — Defines information returned pertaining to a memory region

FILE

<mach_debug/vm_info.h>

SYNOPSIS

struct **vm_info_region** {
vm_offset_t *vir_start*;
vm_offset_t *vir_end*;
vm_offset_t *vir_object*;
vm_offset_t *vir_offset*;
boolean_t *vir_needs_copy*;
vm_prot_t *vir_protection*;
vm_prot_t *vir_max_protection*;
vm_inherit_t *vir_inheritance*;
unsigned int *vir_wired_count*;
unsigned int *vir_user_wired_count*;
};

typedef struct vm_info_region **vm_info_region_t**;

DESCRIPTION

The **vm_info_region** structure defines the information available on a memory region.

FIELDS

vir_start
: Starting addressof region

vir_end
: Ending address of region

vir_object
: Address of the kernel's **vm_object** structure. The structure so located cannot be directly referenced by a user program. This field is provided as a means of identifying the memory object that describes this region.

vir_offset
: Offset into the memory object of the start of this region.

vir_needs_copy
: True if the object needs to be copied

vir_protection
Memory protection code

vir_max_protection
Maximum memory protection code

vir_inheritance
Inheritance attribute

vir_wired_count
Number of times wired

vir_user_wired_count
Number of times user has wired

RELATED INFORMATION

Functions: **mach_vm_region_info**, **vm_region**.

Data Structures: **vm_info_object**.

zone_info

zone_info Structure — Defines zone information for the kernel's use of kernel memory

FILE

<mach_debug/zone_info.h>

SYNOPSIS

struct **zone_info** {
    int *count*;
    vm_size_t *cur_size*;
    vm_size_t *max_size*;
    vm_size_t *elem_size*;
    vm_size_t *alloc_size*;
    boolean_t *pageable*;
    boolean_t *sleepable*;
    boolean_t *exhaustible*;
    boolean_t *collectable*;
};

typedef struct zone_info ***zone_info_array_t**;

DESCRIPTION

The **zone_info** structure defines the information available on the kernel's use of kernel memory.

For related information on VM usage, see the **vm_statistics_data** structure.

FIELDS

count
: Number of elements used now

cur_size
: Current memory utilization

max_size
: How large can this zone grow

elem_size
: Size of an element

alloc_size
: Size used for more memory

pageable
: Zone pageable

sleepable
Sleep if empty

exhaustible
Merely return if empty

collectable
Garbage collect elements

RELATED INFORMATION

Functions: **host_zone_info.**

Data Structures: **zone_name**, **vm_statistics_data**.

zone_name

zone_name Structure — Defines zone names returned by the **host_zone_info** function

FILE

<mach_debug/zone_info.h>

SYNOPSIS

```
#define ZONE_NAME_MAX_LEN 80

struct zone_name {
        char name[ZONE_NAME_MAX_LEN];
};

typedef struct zone_name *zone_name_array_t;
```

DESCRIPTION

The **zone_name** structure defines the names returned for the zones describing the kernel's use of kernel memory.

For related information on vm usage, see the **vm_statistics_data** structure.

FIELDS

name
: Name of zone returned

RELATED INFORMATION

Functions: **zone_info.**

Data Structures: **zone_info**,**vm_statistics_data**.

APPENDIX E Error Return Values

This appendix lists the various kernel return values.

D_ALREADY_OPEN
Exclusive-use device already open

D_DEVICE_DOWN
Device has been shut down

D_INVALID_OPERATION
Bad operation for device

D_INVALID_RECNUM
Invalid record (block) number

D_INVALID_SIZE
Invalid IO size

D_IO_ERROR
Hardware IO error

D_IO_QUEUED

IO queued - do not return result

D_NO_MEMORY

Memory allocation failure

D_NO_SUCH_DEVICE

No such device

D_SUCCESS

Normal device return

D_WOULD_BLOCK

Operation would block, but D_NOWAIT set

EML_BAD_CNT

Invalid syscall number

EML_BAD_TASK

Null task

KERN_ABORTED

The operation was aborted. IPC code will catch this and reflect it as a message error.

KERN_FAILURE

The function could not be performed; a catch-all.

KERN_INVALID_ADDRESS

Specified address is not currently valid.

KERN_INVALID_ARGUMENT

The function requested was not applicable to this type of argument, or an argument

KERN_INVALID_CAPABILITY

The supplied (port) capability is improper.

KERN_INVALID_HOST

Target host isn't actually a host.

KERN_INVALID_NAME

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

KERN_INVALID_RIGHT

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

KERN_INVALID_TASK

Target task isn't an active task.

KERN_INVALID_VALUE

A blatant range error.

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_NAME_EXISTS

The name already denotes a right in the task.

KERN_NO_ACCESS

Bogus access restriction.

KERN_NO_SPACE

The address range specified is already in use, or no address range of the size specified could be found.

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.

KERN_PROTECTION_FAILURE

Specified memory is valid, but does not permit the required forms of access.

KERN_RESOURCE_SHORTAGE

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

KERN_RIGHT_EXISTS

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

KERN_SUCCESS

Successful completion

KERN_UREFS_OVERFLOW

Operation would overflow limit on user-references.

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_SUCCESS

Normal IPC success.

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.

MACH_RCV_BODY_ERROR

Error receiving message body. See special bits.

MACH_RCV_HEADER_ERROR

Error receiving message header. See special bits.

MACH_RCV_IN_SET

Port is a member of a port set.

MACH_RCV_INTERRUPTED
Software interrupt.

MACH_RCV_INVALID_DATA
Bogus message buffer for in-line data.

MACH_RCV_INVALID_NAME
Bogus name for receive port/port-set.

MACH_RCV_INVALID_NOTIFY
Bogus notify port argument.

MACH_RCV_PORT_CHANGED
Port moved into a set during the receive.

MACH_RCV_PORT_DIED
Port/set was sent away/died during receive.

MACH_RCV_TIMED_OUT
Didn’t get a message within the time-out value.

MACH_RCV_TOO_LARGE
Message buffer is not large enough for in-line data.

MACH_SEND_INTERRUPTED
Software interrupt.

MACH_SEND_INVALID_DATA
Bogus in-line data.

MACH_SEND_INVALID_DEST
Bogus destination port.

MACH_SEND_INVALID_HEADER
A field in the header had a bad value.

MACH_SEND_INVALID_MEMORY
Invalid out-of-line memory address.

MACH_SEND_INVALID_NOTIFY
Bogus notify port argument.

MACH_SEND_INVALID_REPLY
Bogus reply port.

MACH_SEND_INVALID_RIGHT
Bogus port rights in the message body.

MACH_SEND_INVALID_TYPE
Invalid msg-type specification.

MACH_SEND_MSG_TOO_SMALL
Data doesn't contain a complete message.

MACH_SEND_NO_BUFFER
No message buffer is available.

MACH_SEND_NO_NOTIFY
Resource shortage; can't request msg-accepted notification.

MACH_SEND_NOTIFY_IN_PROGRESS
Msg-accepted notification already pending.

MACH_SEND_TIMED_OUT
Message not sent before time-out expired.

MACH_SEND_WILL_NOTIFY
Msg-accepted notification will be generated.

MIG_ARRAY_TOO_LARGE
User specified array not large enough to hold returned array

MIG_BAD_ARGUMENTS
Server found wrong arguments

MIG_BAD_ID
Bad message ID

MIG_EXCEPTION
Server raised exception

MIG_NO_REPLY
Server shouldn't reply

MIG_REMOTE_ERROR
Server detected error

MIG_REPLY_MISMATCH
Wrong return message ID

MIG_TYPE_ERROR
Type check failure

APPENDIX F

Function and Data Structure Index