Kernel-Server Loader Functions

To use these functions, you must compile with the kernload library. For example:

```
cc myprog.c -lkernload
```

kern_loader_abort()

SUMMARY Shut down or reconfigure **kern_loader**

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_abort(**port_t *loader_port*, port_t *priv_port*, boolean_t€*restart***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from **kern_loader_look_up()**.

priv_port: The privileged port for this host, returned by **host_priv_self()**.

restart: If true, reconfigure **kern_loader**.

DESCRIPTION This function unloads and deallocates all loadable kernel servers and then, depending on the value of *restart*, kills or reconfigures the kernel-server loader. If *restart* is true, then **kern_loader** rereads its configuration file (**etc/kern_loader.conf**) to determine which servers it should allocate and load.

EXAMPLE

```
/* Get kern_loader's port. */
error=kern_loader_look_up(&loader_port);
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't find kern_loader's port", error);
    exit(1);
}

/* Reconfigure kern_loader. */
error=kern_loader_abort(loader_port, host_priv_self(), TRUE);
if (error != KERN_SUCCESS)
    kern_loader_error("Couldn't stop kern_loader", error);
```

RETURN KERN_SUCCESS: The call was successful.

KERN_LOADER_NO_PERMISSION: *priv_port* isn't the host's privileged port. (Make sure **host_priv_self()** is called by a process with superuser permission.)

SEE ALSO **kern_loader_delete_server()**, **kern_loader_look_up()**, **kern_loader_unload_server()**

kern_loader_add_server()

SUMMARY Allocate a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_add_server(**port_t *loader_port*, port_t *task_port*, server_reloc_t *server_reloc***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

task_port: The kernel's task port, obtained using **task_by_unix_pid()**.

server_reloc: The server's relocatable object file. For example, the relocatable object file of the MIDI driver is ª/usr/lib/kern_loader/Midi/mididriver_relocº.

DESCRIPTION This function prepares the loadable kernel server to be loaded into the kernel. The server isn't loaded unless it automatically loads when allocated.

If the server is already loaded or allocated, then the server is unloaded (if necessary) and allocated again from scratch.

EXAMPLE

```
/* Get kern_loader's port. */
r = kern_loader_look_up(&loader_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get loader_port", r);
    exit(1);
}

/* Get the kernel's task port. */
r = task_by_unix_pid(task_self(), 0, &kernel_task);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get kernel_task", r);
    exit(2);
}

/* Add the server. */
r = kern_loader_add_server(loader_port, kernel_task,
    "/usr/lib/kern_loader/Midi/midi_reloc");
if (r != KERN_SUCCESS) {
    kern_loader_error("Call to kern_loader abort failed", r);
    exit(3);
}
```

RETURN KERN_SUCCESS: The server has been successfully allocated.

KERN_LOADER_NO_PERMISSION: *task_port* wasn't the kernel's task port. (Make sure **task_by_unix_pid()** is called by a process with superuser permission.)

KERN_LOADER_SERVER_WONT_LOAD: The kernel-server loader couldn't use *server_reloc* to build an loadable object file, or it couldn't understand the load or unload commands, or it couldn't link the loadable object file against **/mach**.

SEE ALSO **kern_loader_delete_server()**, **kern_loader_look_up()**

kern_loader_delete_server()

SUMMARY Delete a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_delete_server(**port_t *loader_port*, port_t *task_port*, server_name_t *server_name***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

task_port: The kernel's task port, obtained using **task_by_unix_pid()**.

server_name: The string associated with the server. For example, the name of the MIDI driver is ªmididriverº.

DESCRIPTION This function removes the loadable kernel server from **kern_loader** control. If the server is currently loaded, then it's unloaded.

EXAMPLE

```
/* Get kern_loader's port. */
error=kern_loader_look_up(&loader_port);
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't find kern_loader's port", error);
    exit(1);
}

/* Get the kernel's task port. */
error=task_by_unix_pid(task_self(), 0, &kern_port);
if (error != KERN_SUCCESS) {
    mach_error("Error looking up kernel port", error);
    exit(2);
}

/* Delete the server. */
error=kern_loader_delete_server(loader_port, kern_port, "mididriver");
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't delete mididriver", error);
    exit(3);
}
```

RETURN KERN_SUCCESS: The call succeeded.

KERN_LOADER_NO_PERMISSION: *task_port* wasn't the kernel's task port. (Make sure **task_by_unix_pid()** is called by a process with superuser permission.)

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

SEE ALSO **kern_loader_add_server()**, **kern_loader_look_up()**

kern_loader_error(), kern_loader_error_string()

SUMMARY Display or return an error message

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_error.h>

void **kern_loader_error(**const char **string*, kern_return_t *error***)**
const char ***kern_loader_error_string(**kern_return_t *error***)**

ARGUMENTS *string*: The string to be printed along with the error message.

error: The value returned by a Mach function.

DESCRIPTION These functions act like **mach_error()** and **mach_error_string()**, except that they also understand errors from the kernel-server loader functions.

The **kern_loader_error()** function prints to **stderr** the *string*, followed by the string corresponding to *error*, followed by *error* in parentheses. The **kern_loader_error_string()** function returns the string that corresponds to *error*.

EXAMPLE

```
error=kern_loader_delete_server(loader_port, kern_port, "mididriver");
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't delete mididriver", error);
    exit(3);
}
```

SEE ALSO **mach_error()**, **mach_error_string()**

kern_loader_get_log()

SUMMARY Request a message containing kernel log data

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_types.h>

kern_return_t **kern_loader_get_log(**port_t *loader_port*, port_t *server_com_port*, port_t€*reply_port***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from **kern_loader_look_up()**.

server_com_port: The loadable kernel server's communication port, obtained from **kern_loader_server_com_port()**.

reply_port: The port to which **kern_loader** should send the reply message.

DESCRIPTION This function requests a reply message containing data logged by a loadable kernel server. Before calling this function for the first time on a server, you should turn the server's logging on by calling **kern_loader_log_level()**.

You must supply the implementation of the reply message, as described in Chapter 3, ªUsing Loadable Kernel Servers.º

Each item of logged data is preceded by a time stamp. The time stamp is a relative indicator of when the data was logged by the loadable kernel server.

EXAMPLE

```
r = kern_loader_look_up(&kl_port);
if (r != KERN_SUCCESS) {
    mach_error("Can't find kernel loader", r);
    exit(1);
}

r = port_allocate(task_self(), &reply_port);
if (r != KERN_SUCCESS) {
    mach_error("Can't allocate reply port", r);
    exit(1);
}

/* Get the server's communication port. */
r = task_by_unix_pid(task_self(), 0, &kern_port);
if (r != KERN_SUCCESS) {
    mach_error("Error looking up kernel's port", r);
    exit(1);
}
r = kern_loader_server_com_port(kl_port, kern_port, MYDRIVER_NAME,
    &server_com_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Error looking up server com port", r);
    exit(1);
}

/* Set the log level so we'll get log messages. */
r = kern_loader_log_level(kl_port, server_com_port, LOG_NOTICE);
if (r != KERN_SUCCESS) {
    kern_loader_error("Can't change log level", r);
    exit(1);
}

/* Get the first log message. */
r = kern_loader_get_log(kl_port, server_com_port, reply_port);
if (r != KERN_SUCCESS) {
```

```
        kern_loader_error("Error calling kern_loader_get_log", r);
        exit(1);
    }

    /* Listen for the asynchronous reply message. */
    listen(reply_port);
}

kern_loader_reply_t kern_loader_reply = {
    0,              /* argument to pass to function */
    0,              /* timeout for rpc return msg_send */
    0,              /* string function */
    0,              /* ping function */
    log_data        /* log_data function */
};

void listen(port_name_t port)
{
    char          msg_buf[kern_loader_replyMaxRequestSize];
    msg_header_t *msg = (msg_header_t *)msg_buf;
    kern_return_t r;

    while (1) {
        /* Receive the next message in the queue. */
        msg->msg_size = kern_loader_replyMaxRequestSize;
        msg->msg_local_port = port;
        r = msg_receive(msg, MSG_OPTION_NONE, 0);

        if (r != KERN_SUCCESS) {
            mach_error("listen msg_receive", r);
            exit(1);
        }

        /* Handle the message we just received. */
        kern_loader_reply_handler(msg, &kern_loader_reply);
    }
}

kern_return_t log_data(void *arg, printf_data_t log_data, unsigned int log_data_count)
{
    kern_return_t r;

    /* Print the string we were passed, with our prefix. */
    printf("log_data: %s", log_data);

    /* Deallocate the memory used for the string. */
    vm_deallocate(task_self(), (vm_address_t)log_data,
        log_data_count*sizeof(*log_data));

    /* Get another log message. */
    r = kern_loader_get_log(kl_port, server_com_port, reply_port);
    if (r != KERN_SUCCESS) {
        kern_loader_error("Error calling kern_loader_get_log", r);
        exit(1);
    }
    return KERN_SUCCESS;
}
```

RETURN KERN_SUCCESS: The call succeeded.

KERN_LOADER_UNKNOWN_SERVER: The server is either unknown or has been€deallocated.

KERN_LOADER_SERVER_UNLOADED: The server is only allocated, not loaded.

KERN_LOADER_PORT_EXISTS: Someone is already receiving log messages for this€server.

SEE ALSO **kern_loader_log_level()**, **kern_loader_look_up()**, **kern_loader_reply_handler()**,

kern_loader_server_com_port()

kern_loader_load_server()

SUMMARY Load a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_load_server(**port_t *loader_port*, server_name_t€*server_name*)

ARGUMENTS *loader_port*: **kern_loader** port, obtained from **kern_loader_look_up()**.

server_name: The string associated with the server. For example, the MIDI driver's name is ªmididriverº.

DESCRIPTION This function loads a loadable kernel server that has already been allocated. If the server's relocatable object file has changed since allocation, then the server is allocated again from scratch. This function has no effect on servers that are already loaded; it simply returns KERN_SUCCESS.

EXAMPLE

```
/* Get kern_loader's port. */
error=kern_loader_look_up(&loader_port);
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't find kern_loader's port", error);
    exit(1);
}

/* Load the server. */
error=kern_loader_load_server(loader_port, "mididriver");
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't load the server", error);
    exit(2);
}
```

RETURN KERN_SUCCESS: The server was successfully loaded.

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

KERN_LOADER_SERVER_WONT_LOAD: The server couldn't be loaded.

SEE ALSO **kern_loader_look_up()**, **kern_loader_unload_server()**

kern_loader_log_level()

SUMMARY Set the level of data being logged by a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_log_level(**port_t *loader_port*, port_t *server_com_port*, int€*log_level*)

ARGUMENTS *loader_port*: **kern_loader** port, obtained from **kern_loader_look_up()**.

server_com_port: The loadable kernel server's communication port, obtained from **kern_loader_server_com_port()**.

log_level: An integer indicating the minimum priority of data to be logged. A value of zero turns logging off.

DESCRIPTION This function determines which data logged by a loadable kernel server gets kept. When the server is

first loaded, none of its log messages are kept since its log level is initialized to zero. If you set *log_level* to a value greater than zero, then messages logged at a priority equal to or higher than *log_level* are kept. If you reset the log level to zero, no more log messages are kept until the log level is once again set to a positive value.

Each server can have its own conventions for log priorities.

EXAMPLE

```
r = kern_loader_look_up(&kl_port);
if (r != KERN_SUCCESS) {
    mach_error("Can't find kernel loader", r);
    exit(1);
}

/* Get the server's communication port. */
r = task_by_unix_pid(task_self(), 0, &kern_port);
if (r != KERN_SUCCESS) {
    mach_error("Error looking up kernel's port", r);
    exit(1);
}
r = kern_loader_server_com_port(kl_port, kern_port, MYDRIVER_NAME,
    &server_com_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Error looking up server com port", r);
    exit(1);
}

/* Set the log level so we'll get log messages. */
r = kern_loader_log_level(kl_port, server_com_port, LOG_NOTICE);
if (r != KERN_SUCCESS) {
    kern_loader_error("Can't change log level", r);
    exit(1);
}
```

RETURN KERN_SUCCESS: The log level was successfully set.

KERN_LOADER_UNKNOWN_SERVER: *server_com_port* wasn't valid.

KERN_LOADER_SERVER_UNLOADED: The server isn't currently loaded.

SEE ALSO **kern_loader_look_up()**, **kern_loader_server_com_port()**, **kern_loader_log_level()**

kern_loader_look_up()

SUMMARY Get the **kern_loader** port

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_types.h>

kern_return_t **kern_loader_look_up(**port_t ***loader_port*)**

ARGUMENTS *loader_port*: Returns the port on which **kern_loader** receives messages.

DESCRIPTION This function returns the service port for the kernel-server loader.

EXAMPLE

```
/* Get kern_loader's port. */
error=kern_loader_look_up(&loader_port);
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't find kern_loader's port", error);
    exit(1);
}
```

RETURN KERN_SUCCESS: The call succeeded.

SEE ALSO **kern_loader_server_com_port()**

kern_loader_ping()

SUMMARY Request a synchronization message

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_types.h>

kern_return_t **kern_loader_ping(**port_t *loader_port*, port_t *ping_port*, int *id***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

ping_port: The port to which the ping message should be sent.

id: A value to be sent in the message. You can use this as you wish.

DESCRIPTION You can use this function to make sure that all outstanding status messages have been sent to your program. For example, if you call **kern_loader_status_port()**, you might want to call **kern_loader_ping()** at some point afterward. When you receive the ping message, you'll know that you've received all status messages that were queued before you called **kern_loader_ping()**.

Another reason to call **kern_loader_ping()** is to check whether **kern_loader** has fallen into an unresponsive state.

You must implement the ping message yourself, as described in Chapter 3. The **kern_loader_ping()** function returns a value indicating whether the message was successfully sent to *ping_port*.

EXAMPLE

```
        r = kern_loader_look_up(&kl_port);
    if (r != KERN_SUCCESS) {
        mach_error("kl_util: can't find kernel loader", r);
        exit(1);
    }

    r = port_allocate(task_self(), &reply_port);
    if (r != KERN_SUCCESS) {
        mach_error("kl_util: can't allocate reply port", r);
        exit(1);
    }

    /* Create a thread to listen on reply_port. */
    cthread_detach(cthread_fork((cthread_fn_t)ping_thread,
        (any_t)reply_port));

    /* . . . */

    /* Get a ping message sent to the reply port. */
    r=kern_loader_ping(kl_port, reply_port, 0);

    /* Wait for ping() to kill us.  Exit if we receive a signal. */
    pause();
    exit(0);
}

kern_loader_reply_t kern_loader_reply = {
    0,      /* argument to pass to function */
    0,      /* timeout for rpc return msg_send */
    0,      /* string function */
    ping,   /* ping function */
    0       /* log_data function */
};

void ping_thread(port_name_t port)
```

```
{
    char          msg_buf[kern_loader_replyMaxRequestSize];
    msg_header_t *msg = (msg_header_t *)msg_buf;
    kern_return_t r;

    /* message handling loop */
    while (TRUE) {
        /* Receive the next message in the queue. */
        msg->msg_size = kern_loader_replyMaxRequestSize;
        msg->msg_local_port = port;
        r = msg_receive(msg, MSG_OPTION_NONE, 0);
        if (r != KERN_SUCCESS)
            break;

        /* Handle the message we just received. */
        kern_loader_reply_handler(msg, &kern_loader_reply);
    }

    /* We get here only if msg_receive returned an error. */
    mach_error("ping_thread receive", r);
    exit(1);
}

/* This function is called after a kern_loader_ping. */
kern_return_t ping (void *arg, int id)
{
    exit(0);    /* Kill this process. */
}
```

SEE ALSO **kern_loader_reply_handler()**

kern_loader_reply_handler()

SUMMARY Handle a message from the kernel-server loader

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_reply_handler.h>

kern_return_t **kern_loader_reply_handler(**msg_header_t **msg*, kern_loader_reply_t€**kern_loader_reply***)**

ARGUMENTS *msg*: The message you just received from the kernel-server loader.

kern_loader_reply: A pointer to the structure that specifies which of your functions handle each type of reply from the kernel-server loader.

DESCRIPTION You must use this function if you use **kern_loader_ping()**, **kern_loader_get_log()**, or **kern_loader_status_port()**. Those routines cause an asynchronous reply message from the kernel-server loader; this reply message must be passed to **kern_loader_reply_handler()**, which forwards the message data to your *ping_func()*, *log_func()*, or *string_func()* function.

This function returns the value that is returned by your *ping_func()*, *log_func()*, or *string_func()* function. See Chapter 3 for more information on implementing these functions.

EXAMPLE

```
kern_loader_reply_t kern_loader_reply = {
    0,              /* argument to pass to function */
    0,              /* timeout for rpc return msg_send */
    0,              /* string function */
    0,              /* ping function */
    log_data        /* log_data function */
};

void listen(port_name_t port)
```

```
{
    char          msg_buf[kern_loader_replyMaxRequestSize];
   msg_header_t *msg = (msg_header_t *)msg_buf;
    kern_return_t r;

    while (1) {
        /* Receive the next message in the queue. */
        msg->msg_size = kern_loader_replyMaxRequestSize;
        msg->msg_local_port = port;
        r = msg_receive(msg, MSG_OPTION_NONE, 0);

        if (r != KERN_SUCCESS) {
            mach_error("listen msg_receive", r);
            exit(1);
        }

        /* Handle the message we just received. */
        kern_loader_reply_handler(msg, &kern_loader_reply);
    }
}
```

SEE ALSO **kern_loader_get_log()**, **kern_loader_ping()**, **kern_loader_status_port()**

kern_loader_server_com_port()

SUMMARY Get a loadable kernel server's communication port

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_server_com_port(**port_t *loader_port*, port_t *task_port*, server_name_t *server_name*, port_t ***server_com_port*)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

task_port: The kernel port for the task in which the loadable kernel server is executing. This is returned by **kern_loader_server_task_port()**.

server_name: The string associated with the server. For example, the name of the MIDI driver is ªmididriverº.

server_com_port: Returns the server's communication port.

DESCRIPTION A loadable kernel server's communication port is used for logging-related functions, such as **kern_loader_get_log()** and **kern_loader_log_level()**.

EXAMPLE

```
/* Get kern_loader's port. */
r = kern_loader_look_up(&kl_port);
if (r != KERN_SUCCESS) {
    mach_error("Can't find kernel loader", r);
    exit(1);
}

/* Get the kernel's task port. */
r = task_by_unix_pid(task_self(), 0, &kern_port);
if (r != KERN_SUCCESS) {
    mach_error("Error looking up kernel's port", r);
    exit(1);
}

/* Get the server's com port. */
r = kern_loader_server_com_port(kl_port, kern_port, MYDRIVER_NAME,
    &server_com_port);
if (r != KERN_SUCCESS) {
```

```
    kern_loader_error("Error looking up server com port", r);
    exit(1);
}
```

RETURN KERN_SUCCESS: The call succeeded.

KERN_LOADER_NO_PERMISSION: *task_port* wasn't the server's task port.

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

SEE ALSO **kern_loader_get_log()**, **kern_loader_log_level()**, **kern_loader_look_up()**

kern_loader_server_info()

SUMMARY Get information about a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader_reply.h>

kern_return_t **kern_loader_server_info(**port_t *loader_port*, port_t *task_port*, server_name_t *server_name*, server_state_t **server_state*, vm_address_t€**load_address*, vm_size_t **load_size*, server_reloc_t *relocatable*, server_reloc_t *loadable*, port_name_array_t **port_list*, unsigned€int€**port_list_count*, port_name_string_array_t **port_names*, unsigned€int **port_names_count*, boolean_array_t **advertised*, unsigned€int€**advertised_count***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

task_port: The kernel's task port, obtained using **task_by_unix_pid()**. Specify PORT_NULL if you don't want to have data returned in *port_list*.

server_name: The string associated with the server. For example, the name of the MIDI driver is ªmididriverº.

server_state: Returns the state of the loadable kernel server. The value is one of the following: Zombie, Allocating, Allocated, Loading, Loaded, Unloading, Deallocated (as defined in the header file **kernserv/kern_loader_types.h**).

load_address: Returns the address in the kernel address space where the server starts.

load_size: Returns the number of bytes used by the server. *load_address*€+€*load_size* **-** 1 is the last address in the kernel map that's used by the server's text and data.

relocatable: Returns the location of the relocatable object file for the server.

loadable: Returns the location of the loadable object file (if any) for this server. This is a file created by **kern_loader** from *relocatable* and then loaded against **/mach**.

port_list: Returns the ports that the server has made available to **kern_loader**, using the HMAP or SMAP load command. If you don't pass in the correct *task_port*, this list will consist of null ports.

port_list_count: Returns the number of ports that the server has made available to **kern_loader**. Even if *task_port* isn't valid, and nothing is returned in *port_list*, this argument holds the number of ports that would have been returned.

port_names: Returns the strings associated with the ports in *port_list*.

port_names_count: Returns the number of names in *port_names*. This number is the same as *port_list_count*.

advertised: For each entry in *port_list*, returns true if the port is advertised with the Network Name Server.

advertised_count: Returns the number of entries in *advertised*. This number is the same as *port_list_count*.

DESCRIPTION **kern_loader_server_info()** returns information about a particular loadable kernel server.

EXAMPLE

```
/* Get kern_loader's port. */
error=kern_loader_look_up(&loader_port);
if (error != KERN_SUCCESS) {
    kern_loader_error("Couldn't find kern_loader's port", error);
    exit(1);
}

/* Get the information. */
error=kern_loader_server_info(loader_port, PORT_NULL, "mididriver",
    &server_state, &load_addr, &load_size, relocatable, loadable,
    (port_name_array_t *)&scratch, &count, &port_names, &count,
    &advertised, &count);
if (error != KERN_SUCCESS)
    kern_loader_error("Couldn't get info on mididriver", error);
else
    printf("The relocatable object file is located at:  %s\n",
    relocatable);
```

RETURN KERN_SUCCESS: The call succeeded.

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

SEE ALSO **kern_loader_look_up()**, **kern_loader_server_list()**

kern_loader_server_list()

SUMMARY Get the names of all known loadable kernel servers

SYNOPSIS **#import <mach/mach.h>**

#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_server_list(**port_t *loader_port*, server_name_array_t€**server_names*, unsigned int **server_names_count***)*

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

server_names: Returns an array whose entries are the strings associated with all known€servers.

server_names_count: Returns the number of entries in *server_names*.

DESCRIPTION Use this function to get the string associated with each loadable kernel server that **kern_loader** is keeping track of.

EXAMPLE

```
r = kern_loader_look_up(&loader_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get loader_port", r);
    exit(1);
}

r = kern_loader_server_list(loader_port, &server_names, &count);
if (r != KERN_SUCCESS)
    kern_loader_error("Couldn't get the list", r);
else
    for (i=0; i<count; i++)
        printf("Server %d:  %s\n", i, server_names[i]);
```

RETURN KERN_SUCCESS: The call succeeded.

SEE ALSO **kern_loader_look_up()**, **kern_loader_server_info()**

kern_loader_server_task_port()

SUMMARY Get the task port of a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_server_task_port(**port_t *loader_port*, port_t *kernel_port*, server_name_t *server_name*, port_t ***server_task_port***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

kernel_port: The kernel's task port.

server_name: The string associated with a loaded server. For example, the name of the MIDI driver is ªmididriverº.

server_task_port: Returns the kernel port for the task in which the loadable kernel server is executing.

DESCRIPTION This function returns the task port of the server. Each loadable kernel server currently executes in its own task, but uses the kernel address space. The port returned by **kern_loader_server_task_port()** isn't necessary for any other kernel-server loader functions, but might be useful for gathering debugging information.

EXAMPLE

```
port_t              loader_port, kernel_task, server_port;
kern_return_t       r;
port_name_array_t   names;
unsigned int        i, names_count, types_count;
port_type_array_t   types;

r = kern_loader_look_up(&loader_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get loader_port", r);
    exit(1);
}

r = task_by_unix_pid(task_self(), 0, &kernel_task);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get kernel_task", r);
    exit(2);
}

r = kern_loader_server_task_port(loader_port, kernel_task, "mididriver",
    &server_port);
if (r != KERN_SUCCESS)
    kern_loader_error("Couldn't get the server port", r);
else
    printf("Midi's task port is %d\n", server_port);

r = port_names((task_t)server_port, &names, &names_count, &types,
    &types_count);
if (r != KERN_SUCCESS)
    mach_error("Error calling port_names()", r);
else
    for (i=0; i<names_count; i++)
        printf("Port %d has type %d\n", names[i], types[i]);
```

RETURN KERN_SUCCESS: The call succeeded.

KERN_LOADER_NO_PERMISSION: *task_port* wasn't the server's task port.

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

SEE ALSO **kern_loader_look_up()**, **kern_loader_server_com_port()**

kern_loader_status_port()

SUMMARY Specify a port for **kern_loader** to send status to

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_status_port(**port_t *loader_port*, port_t *listen_port***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

listen_port: The port we want to receive the status messages on.

DESCRIPTION Use this function to get general status from **kern_loader**. You can receive many reply messages as the result of just one call to **kern_loader_status_port()**.

You must define the function that handles status reply messages, as described in Chapter 3. This function receives the status string along with its priority, using the priorities defined in the header file **sys/syslog.h** (LOG_EMERG, LOG_ALERT, and so on).

EXAMPLE

```
        r = kern_loader_look_up(&kl_port);
    if (r != KERN_SUCCESS) {
        mach_error("kl_util: can't find kernel loader", r);
        exit(1);
    }

    r = port_allocate(task_self(), &status_port);
    if (r != KERN_SUCCESS) {
        mach_error("kl_util: can't allocate reply port", r);
        exit(1);
    }

    /* Get generic status messages on this port. */
    r = kern_loader_status_port(kl_port, status_port);
    if (r != KERN_SUCCESS) {
        kern_loader_error("Couldn't specify status port", r);
            exit(1);
    }

    /* Create a thread to listen on status_port. */
    cthread_detach(cthread_fork((cthread_fn_t)receive_thread,
        (any_t)status_port));

    /*
     * Sleep for a while so we can enter kl_util commands at a shell
     * window.  The output of all commands (except status lines from
     * kl_util -s) will show up in both the window that's running this
     * program and in the window that's running kl_util.  (kl_util
     * also has a status port registered.)
     */
    sleep(30);

    exit(0);
}

kern_loader_reply_t kern_loader_reply = {
    0,              /* argument to pass to function */
    0,              /* timeout for rpc return msg_send */
    print_string,   /* string function */
    0,              /* reply_ping function */
    0               /* log_data function */
```

```
};

void receive_thread(port_name_t port)
{
    char          msg_buf[kern_loader_replyMaxRequestSize];
    msg_header_t *msg = (msg_header_t *)msg_buf;
    kern_return_t r;

    /* message handling loop */
    while (TRUE) {
        /* Receive the next message in the queue. */
        msg->msg_size = kern_loader_replyMaxRequestSize;
        msg->msg_local_port = port;
        r = msg_receive(msg, MSG_OPTION_NONE, 0);
        if (r != KERN_SUCCESS)
            break;

        /* Handle the message we just received. */
        kern_loader_reply_handler(msg, &kern_loader_reply);
    }

    /* We get here only if msg_receive returned an error. */
    mach_error("receive_thread receive", r);
    exit(1);
}

/* Called every time kern_loader has status to report. */
kern_return_t print_string(void *arg, printf_data_t string,
    u_int string_count, int level)
{
    /* If the string is empty, return. */
    if (string_count == 0 || !string)
        return KERN_SUCCESS;

    /* Print the string we were passed, with our prefix. */
    printf("print_string: %s", string);

    return KERN_SUCCESS;
}
```

RETURN KERN_SUCCESS: The call succeeded.

SEND_INVALID_PORT: *listen_port* isn't a valid port.

SEE ALSO **kern_loader_look_up()**, **kern_loader_reply_handler()**

kern_loader_unload_server()

SUMMARY Unload a loadable kernel server

SYNOPSIS **#import <mach/mach.h>**
#import <kernserv/kern_loader.h>

kern_return_t **kern_loader_unload_server(**port_t *loader_port*, port_t *task_port*, server_name_t *server_name***)**

ARGUMENTS *loader_port*: **kern_loader** port, obtained from calling **kern_loader_look_up()**.

task_port: The kernel's task port, obtained using **task_by_unix_pid()**.

server_name: The string associated with the server. For example, the name of the MIDI driver is ªmididriverº.

DESCRIPTION Use this function to unload a running loadable kernel server, leaving it allocated.

EXAMPLE

```
r = kern_loader_look_up(&loader_port);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get loader_port", r);
    exit(1);
}

r = task_by_unix_pid(task_self(), 0, &kernel_task);
if (r != KERN_SUCCESS) {
    kern_loader_error("Couldn't get kernel_task", r);
    exit(2);
}

r = kern_loader_unload_server(loader_port, kernel_task,
    "NextDimension");
if (r != KERN_SUCCESS)
    kern_loader_error("Couldn't unload the server", r);
```

RETURN KERN_SUCCESS: The server was successfully unloaded.

KERN_LOADER_SERVER_UNLOADED: The server was already unloaded.

KERN_LOADER_NO_PERMISSION: *task_port* wasn't the kernel's task port. (Make sure **task_by_unix_pid()** is called by a process with superuser permission.)

KERN_LOADER_UNKNOWN_SERVER: *server_name* wasn't recognized.

SEE ALSO **kern_loader_load_server()**, **kern_loader_look_up()**