A

Utilities for Loadable Kernel€Servers

This appendix describes the syntax of several of the commands that operate on loadable kernel servers:

- The kernel-server loader, **kern_loader**, is the program that loads servers into the kernel. It's also discussed in Chapter 5, ªOverview of Loadable Kernel€Servers.º
- The kernel-server linker, **kl_ld**, links object files and command scripts into a relocatable object file. The format of command scripts is discussed in detail in this appendix. An example of using **kl_ld** is in Chapter 9, ªBuilding, Loading, and€Debugging Loadable Kernel€Servers.º
- The kernel-server utility, **kl_util**, is useful for loading and unloading servers, as well as getting information about servers. It's also discussed briefly in Chapter 9.
- The kernel-server log command, **kl_log**, lets you see log messages from a server. Log messages are discussed in Chapter 6, ªDesigning Loadable Kernel Servers.º

Kernel-Server Loader (kern_loader)

The kernel-server loader, **kern_loader**, is the task that adds loadable kernel servers to the kernel. The kernel-server loader works by listening to the ports of known loadable kernel servers. When it intercepts a request for a loadable kernel server, it loads the server and initializes it to respond to this request and subsequent requests.

The kernel-server loader also listens on its own port for requests made through the kernel-server loader functions, a group of user-level functions. The kernel-server loader functions can be used to add and delete known servers, to load servers into the kernel and unload running servers from the kernel, and to get status information. You can use the kernel-server utility, **kl_util**, to communicate with the kernel-server loader (as described later in this appendix), or you can write your own program using the kernel-server loader functions. The kernel-server loader functions are documented in Chapter 3, ªUsing Loadable Kernel Servers.º

When invoked, the kernel-server loader reads its configuration file, **/etc/kern_loader.conf**. This file contains a list of relocatable object files, one for each kernel server that is to be prepared for loading into the kernel. Here's a sample **kern_loader.conf** file:

```
/usr/lib/kern_loader/Midi/mididriver_reloc
/usr/local/lib/kern_loader/Mydriver/mydriver_reloc
```

Starting kern_loader

The **kern_loader** daemon is called automatically during system startup. If it's killed, you€normally can't restart it because the Bootstrap Server won't let any process except **mach_init** register the ªserver_loaderº service. However, if you change a couple of lines€in€the Bootstrap Server configuration file and then reboot, the Bootstrap Server will

let you reinvoke **kern_loader** in the future. Specifically, you should change the following lines in **/etc/bootstrap.conf**:

```
services NetMessage . . .;
. . .
server "/usr/etc/kern_loader -n" services server_loader;
```

to the following:

```
services NetMessage . . . server_loader;
. . .
server "/usr/etc/kern_loader -n";
```

After you make that change to **/etc/bootstrap.conf** and reboot, you can reinvoke **kern_loader** at any time, as follows:

/usr/etc/kern_loader [**-d**] [**-n**] [**-v**] [*relocatable ...*]

The command-line options are:

-d	Don't detach from the invoking terminal; stay in the foreground.
-n	Don't fork another process to be **kern_loader**. This is necessary in the Bootstrap Server configuration file because the Bootstrap Server keeps track of all its servers.
-v	Display debugging information.
relocatable ...	The name of one or more relocatable object files to be read (before those listed in **/etc/kern_loader.conf**).

Kernel-Server Linker (kl_ld)

The relocatable object file of your loadable kernel server must contain certain information: the name of your server, which functions to call to initialize the server, the names of message-handling functions, the name of your server's instance variable, and so on. You put this information into the relocatable object file by using **kl_ld** to link your server.

The syntax for using **kl_ld** follows:

kl_ld -n *server_name* **-i** *instance_var* **-l** *load_cmds_file* [**-u€***unload_cmds_file*] [**-d€***loadable_name*] **-o€***output_file input_file ...*

where:

-n *server_name*	Specifies the name of the kernel server. This name is used in calls to the kernel-server loader functions (such as **kern_loader_load_server()**) and in the **kl_util** and **kl_log** command lines.
-i *instance_var*	Specifies the name of the kernel server's instance variable. This variable's structure must start with a field of type **kern_server_t** (defined in the header file **kernserv/kern_server_types.h**).
-l *load_cmds_file*	Specifies the name of the script that contains commands that **kern_loader** must execute when it loads your server. This file is read into the relocatable object file when you create it. If you want to change the load commands, you must recreate the relocatable object file.
-u *unload_cmds_file*	Specifies the name of the script that contains commands that **kern_loader** must execute when it *un*loads your server. Like load commands, unload commands are read into the relocatable object file when you create it. Thus, you must recreate the

relocatable object file if you want to change the unload commands.

-d *loadable_name* — Specifies the pathname of the loadable object file that **kern_loader** creates from the relocatable object file. This pathname can be either absolute or relative to the directory containing the relocatable object file. Use this option to make **kern_loader** put the loadable object file in a place where the GNU source level debugger, GDB, can easily find and use it.

-o *output_file* — Specifies the name of the relocatable object file that is created. The **kern_loader** will later relocate this file against the kernel.

input_file ... — The object files to be linked into the relocatable object file.

The following example shows a makefile that creates a relocatable object file.

Note: On the last line of the command for the ª$(NAME)_relocº target, ª$@º refers to ª$(NAME)_relocº.

```
NAME=slot
OFILES= slot_server.o slot_handler.o
CFLAGS= -g -DKERNEL -DKERNEL_FEATURES -DMACH
.
.
.
$(NAME)_reloc:  $(OFILES) Load_Commands Unload_Commands
                kl_ld -n $(NAME) -i instance -l Load_Commands \
                    -u Unload_Commands -o $@ $(OFILES)

.c.o:
$(CC) $(CFLAGS) -c $*.c -o $*.o
```

Load Commands

The load commands script can have the commands described in this section. The script must have at least one of the following commands: HMAP, SMAP, or START.

ADVERTISE — Specifies the name of a port that is to be allocated and advertised with the Network Name Server. When **kern_loader** receives messages on any advertised port, the kernel server will be loaded into the kernel and initialized. As part of the initialization sequence, receive rights for the advertised port are forwarded to the kernel server. The message will then be forwarded by **kern_loader** to the loaded kernel server.

Syntax: **ADVERTISE** *port*

CALL — Specifies the name of a function to be called with the specified integer argument as part of the server initialization sequence. If the script has multiple CALL commands, they'll be executed in order.

Syntax: **CALL** *function integer*

DETACH — Specifies that the server should never be unloaded. The DETACH command makes **kern_loader** treat any request to unload the server as an error. Requests to deallocate the server will appear to succeed, but although **kern_loader** stops keeping track of it, the server will remain loaded in kernel memory. The DETACH command is necessary for the correctness of some network protocols.

HMAP — Specifies the mapping of a port to a message-handling function in the kernel server. When **kern_loader** receives a message on this port, it calls the function with the global variable or integer argument you specify. This function must have a handler interface, as opposed to a server interface (see Chapter 6, ªDesigning Loadable Kernel Serversº). To advertise this port with the Network Name Server, use the ADVERTISE command, as described previously in this section.

Syntax: **HMAP** *port_name handler_function argument*

PORT_DEATH — Specifies a function in the server to be called when a port death message is received on its behalf.

Syntax: **PORT_DEATH** *function_name*

SMAP — Specifies the mapping of a port to a message-handling function within the kernel server. When **kern_loader** receives a message on this port, it calls the function with the integer argument you specify. This function must have a server interface, as opposed to a handler interface (see Chapter 6). To advertise this port with the Network Name Server, use the ADVERTISE command, as described previously in this section.

Syntax: **SMAP** *port_name server_function integer*

START — Causes the kernel server to be started immediately, rather than waiting for a message to be received on one of its advertised ports. This is most appropriate for kernel servers that don't listen on any ports, or are wired into kernel data structures for nonserver-style access.

Syntax: **START**

WIRE — Causes the text and data of the loaded kernel server to be wired down (memory resident), making the kernel server immune from unexpected page faults. You must use WIRE if any part of your kernel server can be called from an interrupt handler. If you use WIRE, your kernel server is wired down before any other load commands are executed.

Syntax: **WIRE**

Here's an example of a load commands script.

```
CALL  slot_init 0
PORT_DEATH  slot_port_death

# Associate ports with proc/arg
SMAP  slot0 slot_msg 0
SMAP  slot2 slot_msg 1
SMAP  slot4 slot_msg 2
SMAP  slot6 slot_msg 3

# Server contains interrupt handler code, and so must be wired down
WIRE

# Start this server up immediately
START
```

Unload Commands

The unload commands script can have only CALL commands:

CALL — Specifies the name of a function to be called as part of server shutdown. The function will be passed the specified integer.

Syntax: **CALL** *function integer*

Here's an example of an unload commands script.

```
# Termination

CALL   slot_signoff  0
```

Kernel-Server Utility (kl_util)

The kernel-server utility **/usr/etc/kl_util** lets you communicate with the kernel-server loader. Various options allow you to query the kernel loader for the status of all registered kernel servers, load a kernel server into the kernel, and remove one or more kernel servers from the kernel.

The command-line options to **kl_util** are as follows:

-a *server_reloc_file_name ...*
Causes **kern_loader** to allocate resources for the specified kernel server or servers. Each added server will have kernel space allocated for it and will be initialized to load at that location when referenced.

-A Causes **kern_loader** to shut down; all existing kernel servers are unloaded and deallocated, and the running **kern_loader** task exits.

-d *server_name ...*
Causes **kern_loader** to deallocate the specified kernel server or servers; all physical and virtual resources associated with the kernel server are freed.

-l *server_name ...*
Causes **kern_loader** to load the specified kernel server or servers into the kernel. If you don't use this option, loading is normally done either when the kernel server is allocated (if START is specified in the load commands), or when it receives its first message.

-L Causes **kl_util** not to terminate at the end of its operation, so that further **kern_loader** activity can be monitored. As long as **kl_util** is running, anything logged by **kern_loader** is displayed.

-r Causes **kern_loader** to deallocate all its servers and set itself up from scratch by rereading its configuration file. This is similar to specifying the **-A** option and then restarting **kern_loader**, except that **kern_loader** never actually exits.

-s [*server_name ...*]
Causes **kern_loader** to return information about the status of registered kernel servers. If a server name isn't specified, a list of all known servers is displayed. If a server name is specified, detailed information about that server is displayed.

The following example shows the status of the MIDI driver:

```
# /usr/etc/kl_util -s mididriver
SERVER: mididriver
RELOCATABLE: /usr/lib/kern_loader/Midi/mididriver_reloc
STATUS: Allocated at address 0x10f70000 for 0xa000 bytes
PORTS: mididriver(advertised)
#
```

-u *server_name ...*
Causes **kern_loader** to unload the specified kernel server or servers. (Loaded kernel servers remain in the kernel until they're explicitly unloaded.) Unloading the server causes any wired pages to be unwired; thus, this can be used as a mechanism to free up resources in the system when the server is no longer needed.

Kernel-Server Log Command (kl_log)

You can use the kernel-server log command, **kl_log**, to see log messages from a loadable kernel server. You can

also write your own program that calls the **kern_loader_log_level()** and **kern_loader_get_log()** functions to get log messages. The **kern_loader_log_level()** and **kern_loader_get_log()** functions are discussed in Chapter 3, ªUsing Loadable Kernel Servers.º

You must be superuser to call **kl_log**. It has the following syntax:

/usr/etc/kl_log [**-l** *log_level*] *server_name*

where:

server_name Specifies the loadable kernel server for which you're getting or setting log information. This server must already be loaded.

-l *log_level* Specifies the priority of messages that should be kept. By default, the log level is zero, and no log messages are printed. By setting *log_level* to a positive value, you ensure that log messages from the server will be printed to **stdout** if they have a priority equal to or greater than *log_level*.

You might use **kl_log** as follows:

```
slave# kl_log -l 1 mydriver&
.
.
.
slave# kl_log -l 0 mydriver
slave# jobs
[1]  + Running              kl_log -l 1 mydriver
slave# kill %1
slave#
```

Before you stop collecting messages from a kernel server, you should shut off logging by setting its log level to zero. If you don't set the log level to zero, log messages will accumulate even though no process is collecting the messages.