5

Overview of Loadable Kernel€Servers

This chapter should help you decide whether you need to write a loadable kernel server, as well as give you the background information you need to begin writing one. A *loadable kernel server* is any code that's added to the NeXT™Mach kernel after the system has been booted. Loadable kernel servers are the only way to make third-party code execute within the kernel, as opposed to executing at *user level*.

This chapter first discusses the advantages and disadvantages of loadable kernel servers, and then describes the types of loadable kernel servers that you can write. So that you don't unnecessarily write a loadable kernel server, a list of existing NeXT device drivers is presented. Next, this chapter describes in general how loadable kernel servers work. Finally, it describes the hardware and documentation that you'll need if you write a server.

Advantages and Disadvantages of Loadable Kernel€Servers

Most of the code that runs on a NeXT computer executes at user level. For example, all NEXTSTEP™applications, such as Edit and Digital Librarian™, execute at user level, although they occasionally request services from the kernel. Command-line utilities, such as **make**, also execute at user level. Even system-wide resources such as the Window Server, with its Display PostScript®interpreter, run at user level. User-level programs have several advantages over kernel code:

- Debugging them is much easier than debugging kernel code. For example, running a debugger on kernel code requires two computers and frequently freezes the operating system of the computer that's being debugged.
- Errors in user-level code, unlike those in kernel code, don't cause the computer to *panic* (a state in which the operating system stops running; it can cause data loss and can be corrected only by rebooting).
- User-level programs usually don't decrease the overall performance of the system as much as if they were executed in the kernel. For example, code executed in the kernel causes the kernel to reserve more physical memory, reducing the memory that can be used for user-level programs.

However, sometimes it's necessary to execute code in the kernel, especially when:

- User-level code just can't do the job. For example, the NeXT networking architecture currently requires that network protocols be executed inside the kernel.
- User-level code can't do the job quickly enough. For example, if you have a NeXTbus™device, you can either write a user-level program that calls the slot driver (a driver provided in the NeXTbus Development Kit, NeXT product number N7002) or write your own loadable kernel server to control the device. The performance requirements of the device determine whether it's necessary to write your own loadable kernel server. The NeXTbus interface bus is discussed in Chapter 7, ªNeXTbus Device Drivers.º

Supported Types of Loadable Kernel Servers

Currently, third parties can't write device drivers for non-NeXTbus interfaces, such as SCSI and serial ports. Third parties can write three types of loadable kernel servers:

- Device drivers for NeXTbus boards
- Network protocols
- Network packet sniffers

NeXT computers have several drivers that take care of most interface needs:

Driver	Interface
audio	Sound input and output
fd	Internal floppy disk drive
MIDI	Serial interfaces to MIDI devices
np	NeXT laser printer
od	Internal optical disk
sd	SCSI disk drives of all kinds, including hard disks, floppy disks, and CD-ROMs
sg	Concurrent access to up to four SCSI devices of any kind
slot	NeXTbus devices (this driver is provided in the NeXTbus Development Kit)
sound/DSP	DSP56001 digital signal processor
st	SCSI tape drives
zs	Serial port devices

The fd, np, od, sd, sg, st, and zs drivers are described in the UNIX manual pages, which are available through Digital Librarian. The MIDI and sound/DSP drivers are described in the *NEXTSTEP General Reference*, which is also available through Digital Librarian. The audio driver is accessible only indirectly, through Sound Kit™objects; these objects are described in the *General Reference*.

How Loadable Kernel Servers Work

One of the main features of loadable kernel servers is that they can be loaded into the kernel at any time after the computer has booted. This has several benefits. One is that you can add a server without recompiling the kernel. Another is that a server can be unloaded when it's no longer required, saving memory and other resources. You can also unload and reload a server, which speeds up development because you don't have to reboot every time you change the server.

The *kernel-server loader* (**kern_loader**) is the task that loads and unloads loadable kernel servers. You can communicate with **kern_loader** either with the *kernel-server utility* (**kl_util**, which is described later), or by using the kernel-server loader functions, a group of user-level functions that are described in Chapter 3, ªUsing Loadable Kernel Servers.º

Loadable kernel servers have three states:

- AllocatedÐ**kern_loader** has allocated space and resources for the loadable kernel server and is listening for messages to its ports. However, the server isn't currently running.
- LoadedÐThe loadable kernel server is running. It runs as a thread in a non-kernel task, but with the kernel's address map.
- UnallocatedÐ**kern_loader** has no space or other resources allocated for the loadable kernel server.

Loadable kernel servers can be built with or without Mach messages. A server that's based on messages has the

advantage that it's easily loaded at the moment it's needed. This works because **kern_loader**, when it allocates resources for a server, creates the server's ports and advertises them with the Network Name Server. When a message is received on one of these advertised ports, **kern_loader** loads the allocated server into the kernel and forwards the advertised ports along with the received message to the now loaded and running server.

Loadable kernel servers that don't use messages must be loaded into the system as soon as they're allocated in **kern_loader**. Like other loadable kernel servers, though, they can be allocated at any time and unloaded when they're no longer needed. Since newly loaded servers aren't known to the kernel, those that aren't message-based must add their entry points to the kernel. For example, a UNIX-style server (a server accessed with UNIX system calls) must insert pointers to its entry points into device switch tables.

A loadable kernel server doesn't have to use messages if it's accessed only by table lookups. For example, UNIX-style servers and networking protocols can be implemented without messages.

When started, **kern_loader** reads a configuration file (**/etc/kern_loader.conf**) and allocates the listed kernel servers. (Servers can also be allocated later, using **kl_util** or the kernel-server loader functions.) During allocation, **kern_loader** extracts the following information from the server's object file:

- The name of the loadable kernel server
- The names of its ports, and whether they should be advertised with the Network Name€Server
- Which function should be called when a message is received on a particular port
- Whether the server should be loaded into the kernel immediately or wait for a received€message
- Whether the server should be *wired down* (made memory-resident in kernel virtual€memory)
- Initialization functions to call when the server is loaded
- Shutdown functions to call when the server is unloaded

You specify some or all the information listed above to command scripts that are read by the kernel-server linker, **kl_ld**. Chapter 9, ªBuilding, Loading, and€Debugging Loadable Kernel€Servers,º discusses how to compile and link loadable kernel servers; Appendix A, ªUtilities for Loadable Kernel€Servers,º gives details on **kl_ld** and the format of command scripts.

When **kern_loader** allocates a server, it allocates memory within the Mach kernel and relocates the file against the running kernel's symbol table at the allocated address. The resulting code and data are directly copied into the kernel virtual address space when the server is loaded. While you're debugging your server, you need to specify that **kern_loader** should also save the relocated code and data into a *loadable object file*, which is used by the debugger. (The original object fileÐthe one that **kern_loader** uses to create the loadable object fileÐis called the *relocatable object file*.)

During the loading process, **kern_loader** not only loads the server but also initializes it. Initialization includes associating ports with functions and calling any initialization functions. The server is also wired down if requested.

Before **kern_loader** deallocates a server, it calls the server's shutdown functions. The server is responsible for making sure that it has freed all the kernel resources that it requested (such as dynamically allocated memory).

Before You Start

To write a loadable kernel server, you must be able to program in C. Depending on the kind of loadable kernel server you're writing, you probably don't need to be familiar with UNIX®internals. However, a general background in writing interrupt-driven device drivers or networking protocols is useful.

If you've determined that you need to write a loadable kernel server, you'll need to gather some equipment and documentation.

Hardware You'll Need

To write a server, you'll need the following equipment:

- Two NeXT computers that are running the same major release of NEXTSTEP (for example, NEXTSTEP Release 3).
- A working network connection for both computers.
- A NeXTbus Interface Chip (NBIC) on the CPU board of the test computer (only if you're writing a driver for a NeXTbus device). The NBIC is already installed in every NeXTcube™computer and 68040 Upgrade Board.
- The hardware (if any) your server will control, installed in or connected to one of the€computers.

See Chapter 9 for more information on setting up the two computers.

Documentation You'll Need

Depending on the type of server you're writing, you'll need some or all of the documentation listed in this section. You'll also need the server examples that are under **/NextLibrary/Documentation/NextDev/Examples/ServerVsHandler**.

Part 1 of this manual, ªThe Mach Operating System,º is useful for its description of the NeXT Mach operating system and of Mach messaging, as well as documentation of Mach€functions.

You should also have *NEXTSTEP Development Tools*, which is part of the *NEXTSTEP Developer's Library*. It describes the compiler and debugger that you'll use.

You might need to have the *NEXTSTEP Network and System Administration* manual on hand, especially when you're setting up your computers.

If you're writing a driver for a NeXTbus board, you'll need the hardware specifications and installation guide. These are all part of the NeXTbus Development Kit.

- *NeXTbus Interface Chip Specification*
- *NeXTbus Specification*
- *NeXTbus Development Kit Installation Guide*

The following book has good information for any driver writer. It also has specific information on UNIX drivers.

> *Writing a UNIX Device Driver*. Janet I. Egan and Thomas J. Teixeira. John Wiley & Sons, Inc., 1988.

If you're writing a server with a UNIX-style interface, you might need the information in the following book.

> *The Design and Implementation of the 4.3BSD UNIX Operating System*. Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. Addison-Wesley, 1989.