What Is the Kernel?

Before we get ahead of ourselves, let's take a look at the kernel's role in the operating system. In most operating systems, including UNIX and Windows, the kernel is special software that controls various extremely important elements of the machine. As illustrated in Figure 8.1, the kernel sits between individual running programs and the hardware itself. Performing various critical housekeeping functions for the operating system and acting as a liaison between user-level programs and the hardware, the kernel serves a critical role. Many kernels, including those found in UNIX and Windows systems, include the following core features:

- *Process and thread control.* The kernel dictates which programs run and when they run by creating various processes and threads within those processes. A process is nothing more than some memory allocated to a running program, and the threads are individual streams of execution within a process. The kernel orchestrates various processes and their threads so that multiple programs can run simultaneously and transparently on the same machine.

- *Interprocess communication control.* When one process needs to send data to another process or the kernel itself, it can utilize various interprocess communication features of most kernels to send signals and data.

- *Memory control.* The kernel allocates memory to running programs, and frees that memory when it is no longer required. This memory control is implemented in the kernel's virtual memory management function, which utilizes physical RAM and hard drive space to store information for running processes.

- *File system control.* The kernel controls all access to the hard drive, abstracting the raw cylinders and sectors of the drive into a file system structure.

- *Other hardware control.* The kernel manages the interface between various hardware elements, such as the keyboard, mouse, video, audio, and network devices so various programs can utilize them for input and output operations.

- *Interrupt control:* When various hardware components of the machine need attention (e.g., a packet arriving on the network interface) or a program encounters an usual event (e.g., division by zero), the kernel is responsible for determining how to handle the resulting interrupts. By taking care of the interrupt itself using kernel code or sending information to a particular process to deal with it, the kernel keeps the system operating smoothly.

Figure 8.1. A high-level view of an operating system kernel and its relationship to user-level processes and hardware.

With these features, the kernel is all about control: sitting at the interstices of user programs and hardware and controlling what happens on the machine.

As it runs, the kernel relies on hardware-level protections implemented in the system's CPU. By using hardware-level protection, the kernel tries to safeguard its own critical data structures from accidental or deliberate manipulation by user-level processes on the machine. Most CPUs include hardware features to let software on the system run at different levels of privilege. The memory space and other elements of highly sensitive software (like the kernel) cannot be accessed by code running at a less-important level (e.g., user processes). On x86-compatible CPUs, these different sensitivity levels are called *rings*, and range from Ring 0, the most sensitive level, to Ring 3, the least sensitive level. As it runs different tasks, the CPU switches between these different levels depending on the sensitivity of the particular software currently executing.

For the Linux and Windows operating systems, only Rings 0 and 3 are used; the other options supported by x86 CPUs (i.e., Rings 1 and 2) are not utilized. The kernel itself, in both Linux and Windows, runs in Ring 0. In fact, running in Ring 0 defines a given task as being at kernel level. If you run in Ring 0, you can access all of the kernel's memory structures, and are therefore at the same level as the kernel code. User mode processes run in Ring 3, and, under most conditions, are not able to access kernel space directly. By relying on Ring 0 and Ring 3, all software on the machine is really carved up into two different worlds: kernel mode (running in Ring 0) and user mode (running in Ring 3). For non-x86 CPUs, operating systems utilize analogous concepts to Ring 0 and Ring 3 implemented in the CPU's hardware. Nearly all CPUs support some notion of a privileged mode, where the kernel lives, and a nonprivileged mode for user processes. Throughout the rest of this chapter, we'll use x86-specific terminology Ring 0 and Ring 3, as it so dominates literature on this topic.

So, your operating system really consists of two worlds: user mode and kernel mode. The user mode is what you typically see and interact with on a day-to-day basis on your system, as it includes the programs you run, such as a command shell, GUI, mail server, or text editor. The other world, kernel mode, lies silently underneath the whole operation managing access to the hardware and generally controlling things. When a system boots up, the kernel is loaded into memory and begins execution in Ring 0, thereby creating the first world (kernel mode). After the kernel gets itself set up in memory, it activates various user-mode processes that allow individual users to access the system and run programs, thereby creating the user-mode world.

It's important to note that kernel mode is a very different concept from root or administrator permissions. When an administrator runs a command, a given program executes within user mode; that is, in Ring 3. From the kernel's perspective, the administrator is just another user, albeit an important one, but still someone living in Ring 3.

When most programs run, control sometimes has to pass from user mode into kernel mode, such as when the program needs to interact with hardware for printing to the screen, receiving a packet, or some other action. When this happens, control is very carefully passed from user mode to kernel mode, through tightly controlled interfaces. The software that implements this

transition from Ring 3 to Ring 0 is referred to as a *call gate*, as it acts as a gate for user-mode processes into software living in kernel mode.

When administrators ask for a list of running programs using tools like the UNIX `ps`, `lsof`, or `top` commands or the Windows Task Manager, they execute a command from user mode, which asks the kernel to list all running processes. The kernel grabs data from its kernel-mode data structures, responds to the user-mode command with the appropriate information, and the running processes are displayed. Similarly, the administrator or users might ask for a list of files in a directory. The kernel responds with the appropriate information. Or, you could look for which TCP or UDP ports are in use, or whether the network interface is in promiscuous mode. You might even run a file integrity checker to see if any of your critical system files have been altered with a user-mode RootKit. All of these interactions, and far more, rely on the kernel to determine the status of the machine. That's how it's all supposed to work. The kernel takes care of business, and everyone is happy.

PREV < Day Day Up > NEXT