CHAPTER 4

Threads

Learning Objectives

After studying this chapter, you should be able to:

- Understand the distinction between process and thread.
- Describe the basic design issues for threads.
- Explain the difference between user-level threads and kernel-level threads.
- Describe the thread management facility in Windows.
- Describe the thread management facility in Solaris.
- Describe the thread management facility in Linux.

This chapter examines some more advanced concepts related to process management, which are found in a number of contemporary operating systems. We show that the concept of process is more complex and subtle than presented so far and in fact embodies two separate and potentially independent concepts: one relating to resource ownership, and another relating to execution. This distinction has led to the development, in many operating systems, of a construct known as the **thread**.

4.1 PROCESSES AND THREADS

The discussion so far has presented the concept of a process as embodying two characteristics:

1. **Resource ownership:** A process includes a virtual address space to hold the process image; recall from Chapter 3 that the process image is the collection of program, data, stack, and attributes defined in the process control block. From time to time, a process may be allocated control or ownership of resources, such as main memory, I/O channels, I/O devices, and files. The OS performs a protection function to prevent unwanted interference between processes with respect to resources.
2. **Scheduling/execution:** The execution of a process follows an execution path (trace) through one or more programs (e.g., Figure 1.5). This execution may be interleaved with that of other processes. Thus, a process has an execution state (Running, Ready, etc.) and a dispatching priority, and is the entity that is scheduled and dispatched by the OS.

Some thought should convince the reader that these two characteristics are independent and could be treated independently by the OS. This is done in a number of operating systems, particularly recently developed systems. To distinguish the two characteristics, the unit of dispatching is usually referred to as a thread or

lightweight process, while the unit of resource ownership is usually referred to as a **process** or **task**.[1]

Multithreading

Multithreading refers to the ability of an OS to support multiple, concurrent paths of execution within a single process. The traditional approach of a single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach. The two arrangements shown in the left half of Figure 4.1 are single-threaded approaches. MS-DOS is an example of an OS that supports a single-user process and a single thread. Other operating systems, such as some variants of UNIX, support multiple user processes, but only support one thread per process. The right half of Figure 4.1 depicts multithreaded approaches. A Java runtime environment is an example of a system of one process with multiple threads. Of interest in this section is the use of multiple processes, each of which supports multiple threads. This approach is taken in Windows, Solaris, and many

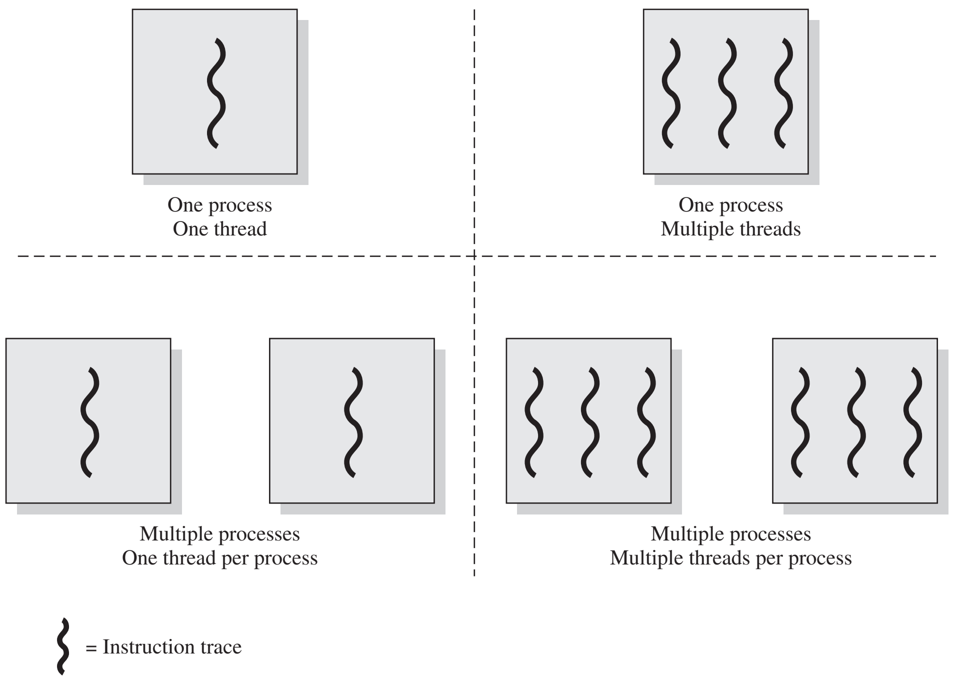

Figure 4.1 Threads and Processes

[1]Alas, even this degree of consistency is not maintained. In IBM's mainframe operating systems, the concepts of address space and task, respectively, correspond roughly to the concepts of process and thread that we describe in this section. Also, in the literature, the term *lightweight process* is used as either (1) equivalent to the term *thread*, (2) a particular type of thread known as a kernel-level thread, or (3) in the case of Solaris, an entity that maps user-level threads to kernel-level threads.