4 UNIX Process Management

Since UNIX is one of multi-user and multiprocessing operating systems, UNIX has its solution to the system resource management. The UNIX kernel handles almost all the basic issues related to process management, memory management, file system, and I/O system, and provide well-defined system programs that have the clear-cut assignment of responsibility in order to allow user programs to call them with system calls (Bach 2006; McKusick et al 2005; Mohay et al 1997). This chapter will discuss the UNIX process management. Chapter 5 will introduce the UNIX memory management. Chapter 6 will focus on the UNIX file system. Chapter 7 will be related to the UNIX I/O system.

In this chapter, multiple processes’ running concurrently will be introduced first. Then, process states, jobs, process and job attributes, and process and job control will be discussed. Foreground and background processes will be explained, too. And finally, UNIX system root and init process will be involved.

4.1 Multiple Processes’ Running Concurrently

In a single-user and single-process operating system, programs must be executed one after another. In this environment, the execution of a program has a significant feature, that is, the execution is unable to break in half way by other programs and resume later without effect, except by the system interruptions. In other words, during the duration from the start to the end of a program execution, the program execution occupies the CPU and memory without sharing them with any other program until its execution finishes. The reason is that these kinds of operating systems do quite little work related to the management of CPU and primary memory. However, the exclusive execution of a program can be quite a waste. For example, when an I/O-intense program is executed, most of time, CPU is idle and just waiting for I/O device operations without doing anything. For most of modern computer systems that usually have much faster microprocessors and much
larger memory space than their ancestors, a single-user and single-program operating system cannot manage well and make the best use of all the system resources. Thus, operating system developers have developed multi-user and multi-processing operating systems.

In a multi-user and multi-processing operating system, not only the executions of several programs can share CPU in an interlaced way without any undesirable impact, but also several users can interact with the system without any delay (Andrews 1983; Bach 2006; Cmelik et al 1989; Sarwar 2006; Sielski 1992; Stallings 1998; Zhou 2009). The reason is that these kinds of operating systems do quite a lot of work on the management of CPU and primary memory in order to make the best use of the system resources. Included in the most important parts of system resource management are process management, memory management, file system, and I/O device system. Among them, process management tackles how to control and schedule CPU to accomplish the tasks of different users successfully.

In a multi-user and multi-processing operating system, the only active entities are the processes. A process is an execution process of a program. In a large multi-user time-sharing system, there may be hundreds or even thousands of processes running, and each user on the system may have several active processes at once. In fact, even though no users use the system that is running, dozens of background processes, called daemons, are executing.

As one of the multi-user and multi-processing operating systems, UNIX has its own solution to the system resource management. UNIX kernel handles almost all the basic problems related to process management, memory management, file system, and I/O device system. UNIX processes are very similar to the classical sequential processes. Each process runs a single program and initially has a single thread of control.

Multiprogramming not only allows the processor to handle multiple batch jobs at a time, but also be used to handle multiple interactive jobs. This is multiple processes’ running concurrently. In a time-sharing system, the processor’s time is shared among multiple users, and the execution of each user program is in a short period of time of computation. This short period of execution time for each user program is called a burst (or quantum, or time slice). In a UNIX operating system, the quantum is usually 0.1 – 1 sec. That is, one process is executed for a quantum, and then the CPU is taken away from it and given to another process. The new process executes for a quantum and then the CPU is given to the next process. The process management part of the UNIX kernel takes the responsibility to schedule and switch context for the next process that is ready to run.

A process has its lifetime, which starts from the moment when UNIX kernel creates it and ends at the time when it is removed from the system. Process creation with the fork system call and termination via the exit system call are the only mechanisms used by the UNIX system to execute external commands and programs. Every time a user runs an external command or program, the UNIX kernel creates a process for it to execute; once its execu-