Early computers allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system’s resources. In contrast, contemporary computer systems allow multiple programs to be loaded into memory and executed concurrently. This evolution required firmer control and more compartmentalization of the various programs; and these needs resulted in the notion of a process, which is a program in execution. A process is the unit of work in a modern computing system.

The more complex the operating system is, the more it is expected to do on behalf of its users. Although its main concern is the execution of user programs, it also needs to take care of various system tasks that are best done in user space, rather than within the kernel. A system therefore consists of a collection of processes, some executing user code, others executing operating system code. Potentially, all these processes can execute concurrently, with the CPU (or CPUs) multiplexed among them. In this chapter, you will read about what processes are, how they are represented in an operating system, and how they work.

**Bibliographical Notes**


Message passing for multicore systems is discussed in [Holland and Seltzer (2011)]. [Baumann et al. (2009)] describe performance issues in shared-memory and message-passing systems. [Levin (2013)] describes message passing in the Mach system, particularly with respect to macOS and iOS.

and [Robbins and Robbins (2003)] cover pipes in Windows and UNIX systems, respectively.

Bibliography


