Advanced Operating Systems Concepts

This note is an expanded version of the “Advanced Operating Systems Concepts” section of Chapter 2, Computer Systems, on pages 41 and 42 of Managing Information Technology, Seventh Edition. It is included on the book’s Web site for those students who wish to obtain a better understanding of the concepts of multiprogramming and virtual memory.

Multiprogramming, Multitasking, or Multithreading

Operating systems often incorporate two important concepts—multiprogramming (or multitasking or multithreading) and virtual memory—in order to increase the efficiency of the computer’s operations. These concepts are concerned with the management of the CPU time and the memory of the computer system.

On larger machines, multiprogramming is often employed to overlap input and output operations with processing time. This is very important because the time required for the computer to perform an input/output operation (such as reading from disk) is quite large compared to the time required to execute an arithmetic instruction. A typical computer might execute 100,000 arithmetic instructions in the time required to read a single record from a disk. Thus, it would be quite inefficient to let the CPU remain idle while input/output operations are being completed. Multiprogramming keeps the CPU busy by overlapping the input/output operations of one program with the processing time of another program.

For multiprogramming, several programs (say 5 to 10) must be located in memory at the same time. Then the operating system supervises the switching back and forth among these programs so that the CPU is almost always busy. When the currently executing program encounters an input/output instruction, an interrupt occurs and the operating system takes control. The operating system stores the status of the interrupted program in memory so that this information will be available when the interrupted program gets another shot at the CPU. The operating system then decides which of the waiting programs should be executed next, and it resets the computer with the new program’s status. Then the operating system gives control to the new program, which executes until it encounters an input/output instruction. Thus, the operating system controls the switching back and forth among programs that is involved in multiprogramming.

The switching among programs in multiprogramming may be triggered by time as well as by an event (the occurrence of an input/output instruction). Time-driven multiprogramming (sometimes called time-sharing) is the usual mode of operation when large numbers of users are simultaneously using a computer (midrange or larger) from terminals or microcomputers serving as terminals. In this environment, each user is allocated a small slice of CPU time (e.g., a few milliseconds). When a particular user’s turn arises, her program runs for those few milliseconds, carrying out thousands of instructions. Then a time interrupt occurs, and the operating system transfers control to the next user for his slice of time. Unless the number of concurrent users becomes excessively high, these bursts of available time occur so rapidly that it appears to the user that he or she is the only person who is using the computer.
On microcomputers, the term *multitasking* is used to describe essentially the same function as multiprogramming on larger machines. In both cases the operating system controls the switching back and forth among programs stored in memory. There are two basic types of multitasking: *preemptive* and *cooperative*. In preemptive multitasking, the operating system allocates slices of CPU time to each program (the same as time-driven multiprogramming previously). In cooperative multitasking, each program can control the CPU for as long as the program needs. In practice, multitasking means that a user can print a report at essentially the same time as he or she recalculates a spreadsheet, all the while monitoring for new electronic mail.

Finally, *multithreading* (*thread* is short for *thread of execution*) is almost the same as multitasking except that the multiple threads are different parts of the *same* program that are being executed near simultaneously, with the operating system controlling the switching back and forth among threads of the single program. All three terms—*multiprogramming*, *multitasking*, and *multithreading*—refer to the efforts of the operating system to maximize the work done by the CPU.

**Virtual Memory**

Whereas multiprogramming or multitasking is primarily concerned with the management of CPU time, *virtual memory* is concerned with the management of main memory. Until the mid 1990s, virtual memory was used only on larger computer systems, but now it is used on microcomputers as well. Virtual memory makes it appear to the user that an unlimited amount of main memory is available, meaning that individual programs can be much larger than the actual number of memory cells. More importantly, virtual memory permits multiprogramming to operate more efficiently. How does this work?

The trick is the creative use of direct access storage devices (DASDs), with the operating system switching portions of programs (called pages) between main memory and DASDs. Unless all the programs are small, it is difficult to get enough programs stored in memory for multiprogramming to operate efficiently. For example, three large programs may occupy all of the memory, and it may be common for all three programs to be processing input/output instructions at the same time. This leaves the CPU idle, which is undesirable. The cost of adding enough real memory to store 10 programs at a time—to permit efficient multiprogramming—might be prohibitive. The virtual memory concept recognizes that only one segment of a large program is being executed at a time, while the bulk of the program is inactive. Therefore, with virtual memory, only a few pages of the program are kept in main memory, with the rest relegated to a DASD. Because only a small portion of each program is located in memory, portions of a sufficient number of programs can be stored in memory to permit efficient multiprogramming.

Of course, it is often necessary for the operating system to bring new portions of a program (new pages) into memory so they can be executed. This swapping of pages between a DASD and main memory is called, appropriately enough, paging. The size of pages varies, but each is often a few thousand bytes. When we combine the concepts of multiprogramming (switching among pages of programs already in memory) with virtual memory (requiring frequent page switches from DASDs to memory), then we begin to realize the incredible complexity of tasks carried out by the operating system.