CHAPTER 18

Now that you understand the fundamental concepts of operating systems (CPU scheduling, memory management, processes, and so on), we are in a position to examine how these concepts have been applied in several older and highly influential operating systems. Some of them (such as the XDS-940 and the THE system) were one-of-a-kind systems; others (such as OS/360) are widely used. The order of presentation highlights the similarities and differences of the systems; it is not strictly chronological or ordered by importance. The serious student of operating systems should be familiar with all these systems.

Exercises

18.1 Discuss what considerations the computer operator took into account in deciding on the sequences in which programs would be run on early computer systems that were manually operated.
Answer:
Jobs with similar needs are batched together and run together to reduce set-up time. For instance, jobs that require the same compiler because they were written in the same language are scheduled together so that the compiler is loaded only once and used on both programs.

18.2 What optimizations were used to minimize the discrepancy between CPU and I/O speeds on early computer systems?
Answer:
An optimization used to minimize the discrepancy between CPU and I/O speeds is spooling. Spooling overlaps the I/O of one job with the computation of other jobs. The spooler for instance could be reading the input of one job while printing the output of a different job or while executing another job.

18.3 Consider the page replacement algorithm used by Atlas. In what ways is it different from the clock algorithm discussed in Section 9.4.5.2?
Answer:
The page replacement algorithm used in Atlas is very different from the clock algorithm discussed in earlier chapters. The Atlas system keeps track of whether a page was accessed in each period of 1024 instructions to a page, while t2 is the interval between the last two references of a page. The paging system then discards any page that has t1 > t2 + 1. If it cannot find any such page, it discards the page with the largest t2 - t1. This algorithm assumes that programs access memory in loops and the idea is to retain pages even if it has not been accessed for a long time if there has been a history of accessing the page regularly albeit at long intervals. The clock algorithm, on the other hand, is an approximate version of the least recently used algorithm and therefore discards the least recently used page without taking into account that some of the pages might be infrequently but repeatedly accessed.

18.4 Consider the multilevel feedback queue used by CTSS and MULTICS. Suppose a program consistently uses seven time units every time it is scheduled before it performs an I/O operation and blocks. How many time units are allocated to this program when it is scheduled for execution at different points in time?
Answer:
Assume that the process is initially scheduled for one time unit. Since the process does not finish by the end of the time quantum, it is moved to a lower level queue and its time quantum is raised to two time units. This process continues till it is moved to a level 4 queue with a time quantum of 8 time units. In certain multilevel systems, when the process executes next and does not use its full time quantum, the process might be moved back to a level 3 queue.

18.5 What are the implications of supporting BSD functionality in user-mode servers within the Mach operating system?

Answer:

Mach operating system supports the BSD functionality in user mode servers. When the user process makes a BSD call, it traps into kernel mode and the kernel copies the arguments to the user level server. A context switch is then made and the user level performs the requested operation and computes the results which are then copied back to the kernel space. Another context switch takes place to the original process which is in kernel mode and the process eventually transitions from kernel mode to user mode along with the results of the BSD call. Therefore, in order to perform the BSD call, there are two kernel crossings and two process switches thereby resulting in a large overhead. This is significantly higher than the cost if the BSD functionality is supported within the kernel.

18.6 What conclusions can be drawn about the evolution of operating systems? What causes some operating systems to gain in popularity and others to fade?

Answer:

Operating systems that have made advances in operating system technology —that is, advances to memory management or interprocess communication — have typically been the types of systems that are both popular and have existed for a period of time. UNIX is a classic example of a type of system that has made significant technological advances and has lasted for more than 30 years. Further evidence of this are the types of systems that have evolved from UNIX and have gained in popularity on their own. Linux is perhaps the most notable example. The types of operating systems that have faded from view are typically systems that are either too specific in purpose or lack in performance. There is much more motivation to replace such systems rather than continuing to advance them.