CHAPTER 5

Process Synchronization

Practice Exercises

5.1 In Section 5.4, we mentioned that disabling interrupts frequently can affect the system's clock. Explain why this can occur and how such effects can be minimized.

Answer:

The system clock is updated at every clock interrupt. If interrupts were disabled—particularly for a long period of time—it is possible the system clock could easily lose the correct time. The system clock is also used for scheduling purposes. For example, the time quantum for a process is expressed as a number of clock ticks. At every clock interrupt, the scheduler determines if the time quantum for the currently running process has expired. If clock interrupts were disabled, the scheduler could not accurately assign time quantums. This effect can be minimized by disabling clock interrupts for only very short periods.

5.2 Explain why Windows, Linux, and Solaris implement multiple locking mechanisms. Describe the circumstances under which they use spinlocks, mutex locks, semaphores, adaptive mutex locks, and condition variables. In each case, explain why the mechanism is needed.

Answer:

These operating systems provide different locking mechanisms depending on the application developers' needs. Spinlocks are useful for multiprocessor systems where a thread can run in a busy-loop (for a short period of time) rather than incurring the overhead of being put in a sleep queue. Mutexes are useful for locking resources. Solaris 2 uses adaptive mutexes, meaning that the mutex is implemented with a spin lock on multiprocessor machines. Semaphores and condition variables are more appropriate tools for synchronization when a resource must be held for a long period of time, since spinning is inefficient for a long duration.

5.3 What is the meaning of the term **busy waiting**? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Answer:
Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

5.4 Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.
Answer:
Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock can be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

5.5 Show that, if the `wait()` and `signal()` semaphore operations are not executed atomically, then mutual exclusion may be violated.
Answer:
A wait operation atomically decrements the value associated with a semaphore. If two wait operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value, thereby violating mutual exclusion.

5.6 Illustrate how a binary semaphore can be used to implement mutual exclusion among n processes.
Answer:
The n processes share a semaphore, `mutex`, initialized to 1. Each process P_i is organized as follows:

```
do {
   wait(mutex);

      /* critical section */

   signal(mutex);

      /* remainder section */
} while (true);
```