Synchronization Examples

Practice Exercises

7.1 Explain why Windows and Linux implement multiple locking mechanisms. Describe the circumstances under which they use spinlocks, mutex locks, semaphores, 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 spinlock 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.

7.2 Windows provides a lightweight synchronization tool called **slim reader–writer** locks. Whereas most implementations of reader–writer locks favor either readers or writers, or perhaps order waiting threads using a FIFO policy, slim reader–writer locks favor neither readers nor writers, nor are waiting threads ordered in a FIFO queue. Explain the benefits of providing such a synchronization tool.

Answer:
Simplicity. If reader–writer locks provide fairness or favor readers or writers, they involve more overhead. Providing such a simple synchronization mechanism makes access to the lock fast. Use of this lock may be most appropriate for situations where reader–writer locks are needed, but quickly acquiring and releasing them is similarly important.

7.3 Describe what changes would be necessary to the producer and consumer processes in Figure 7.1 and Figure 7.2 so that a mutex lock could be used instead of a binary semaphore.

Answer:

The calls to `wait(mutex)` and `signal(mutex)` need to be replaced so that they are now calls to the API for a mutex lock, such as `acquire(mutex)` and `release()` mutex.

7.4 Describe how deadlock is possible with the dining-philosophers problem.

Answer:
If all philosophers simultaneously pick up their left forks, when they turn to pick up their right forks they will realize they are unavailable, and will block while waiting for it to become available. This blocking while waiting for a resource to become available is a deadlocked situation.

7.5 Explain the difference between signaled and non-signaled states with Windows dispatcher objects.

Answer:
An object that is in the signaled state is available, and a thread will not block when it tries to acquire it. When the lock is acquired, it is in the non-signaled state. When the lock is released, it transitions back to the signaled state.

7.6 Assume `val` is an atomic integer in a Linux system. What is the value of `val` after the following operations have been completed?

```
atomic_set(&val,10);
atomic_sub(8,&val);
atomic_inc(&val);
atomic_inc(&val);
atomic_add(6,&val);
atomic_sub(3,&val);
```

Answer:
The final value of `val` is 10 - 8 + 1 + 1 + 6 - 3 = 7