CS 377 – Operating Systems

Discussion Session 5 Questions

Name: _____

Write your answers individually, without consulting notes, slides, books, or the internet. Be succinct (complete sentences not necessary). **Remember to turn your paper over.**

1. **Locks**. Locks are basic OS primitive used to support synchronization between threads.

   (a) What are the two essential operations of a lock and what (briefly) do they do?

   Solution. *Acquire and Release. A thread calls acquire to take control of a lock – if/when it succeeds, then the thread knows that no other thread will be able to take control of the lock until the thread calls release on the lock, which returns it to the pool of available (free) locks. During the period the lock is acquired, the acquiring thread can run operations without worrying about other threads in the same synchronized section.*

   (b) What is an **atomic instruction** (such as `Test&Set`) and why are they needed to support locks (assuming we aren't disabling interrupts)?

   Solution. *Atomic instructions are hardware-supported operations that are guaranteed to never be interrupted – i.e., hardware-supported synchronization primitives. These cannot be implemented in software and so must be provided by the hardware. They are needed because in order to acquire a lock, you must be able to both check that the lock is available and then take it without the possibility that two threads will both perform the check at the same time and pass before either one takes the lock. Having the test and set happen as one step ensures that only one thread will be able to capture the lock.*

   (c) What is **busy waiting** and how can we reduce the amount of time spent doing it?

   Solution. *Busy waiting refers to actively expending CPU cycles repeatedly checking that a lock is still unavailable (that is, performing no useful work but still using CPU time). In the worst case, a thread may consume 100% of the available CPU simply checking the lock value over and over again. We can reduce the time spent busy waiting by having the thread yield the CPU if the lock is busy, then wake it up only when the lock is released (possibly along with other threads). One of the woken threads will then acquire the lock, while the others will again yield.*

2. **Semaphores**. Semaphores are integers used for synchronization that are updated using the Wait and Signal operations.

   (a) How can you implement a lock using a semaphore?

   Solution. *Use a binary semaphore – this is a semaphore that is initialized to 1 and always equals either 0 or 1. Only one thread can capture the 'lock' (by calling Wait, which sets the semaphore to 0), which will then release the lock by calling Signal (setting the semaphore back to 1).*

   (b) How are semaphores more general/powerful than locks?

   Solution. *Since semaphores have the ability to count (unlike locks), they can be used for other purposes as well, such as controlling access to a fixed pool of shared resources (via a counting semaphore) or expressing scheduling constraints (via one thread Waiting for another thread to call Signal).*

3. **Monitors**. A monitor is another synchronization construct that consists of a lock and zero or more **condition variables**. What is a condition variable?

   Solution. *A condition variable is a queue of threads that are waiting for some condition inside a critical section (i.e., a section with mutual exclusion). Condition variables allow giving up the critical section's lock while still within the critical section, and ensuring that the lock is held again when the thread wakes up (once the condition is true).*