Jeremy Holloway
CPSC-3220-001
5/27/2019

Pre-class Assignment #8

1. Under what two conditions does the book say it is appropriate to replace a mutual exclusion lock with a RWLock?

When there is substantial contention (multiple threads are going for the lock at once) for the mutual exclusion lock and a substantial majority of the accesses are read-only.

2. Why does the book's implementation of RWLock::doneWrite() in Figure 5.10 use a broadcast rather than a signal?

Multiple threads can read data at once so it makes sense to wake them all up at once. The function checks if any threads need to write and if not it signals all threads to start reading.

3. Why is a FIFO BBQ (blocking bounded queue) more complex to implement than the BBQ implementation given in Figure 5.8?

The earlier version does not guarantee freedom from starvation, which allows all the threads to make progress.

4. What is the key idea in implementing a FIFO BBQ?

There is an explicit queue with every node having its own queue, which ensures only one thread is waiting on that condition.

5. Under what condition(s) is it valid to use disabling interrupts as a way in which to provide a lock?

When you would need to make a sequence of instructions atomic on a single processor.

6. What does a test_and_set instruction do? Why must it be atomic?

It atomically reads a value from memory to a register and writes a one to that memory location. The memory state could cause other actions in the shared state or have the memory values interfered with.

7. Why does a SpinLock not have a waiting list?

The thread is not blocked it keeps executing.

8. Under what conditions does the textbook say that spin locks make sense?

Spin locks make sense for locks that only need to be held for a short period of time.

9. Why does a multiprocessor queueing lock use both interrupt disabling and a SpinLock?

To ensure the thread is not preempted while holding the ready list spinlock.

10. Why is there a lockAcquire() operation at the bottom of CV::wait() in Figure 5.18?

To prevent a thread from being back out on the ready list before a context switch has been completed.