CS 162 Section 2 Spring 2014

True/False

1. Preemptive multithreading requires threads to give up the CPU using the `yield()` system call

2. In some cases, two threads can both be executing a specific critical section at the same time

3. Every interrupt results in a transition from user to kernel mode. Hint: think Inception

4. A thread needs to own a semaphore, meaning the thread has called `semaphore.P()`, before it can call `semaphore.V()`

5. A thread needs to own the monitor lock before it can `signal()` a condition variable.

Short Answer

1. With spinlocks, threads spin in a loop (busy waiting) until the lock is freed – in general this is a bad idea and wastes many CPU cycles. However, there are certain cases where a spinlock is more efficient than a blocking lock (puts waiting thread to sleep). When?

2. What are some similarities and differences between interrupts and system calls?

Locks

Consider the two spinlock implementations below. Are they correct? If not, give an example scenario that would cause the lock to break.
Assume a system with two threads, and in #2 this_thread and other_thread correspond to integer thread IDs.

1.

```
struct lock {
     int held = 0;
}

void acquire(lock) {
     while(lock->held);
     lock->held = 1;
}

void release(lock) {
     lock->held = 0;
}
```

2.

```
struct lock {
     int turn = 0;
}

void acquire(lock) {
     while(lock->turn != this thread);
}

void release(lock) {
     lock->turn = other_thread;
}
```

Semaphores

Implement the P() and V() methods of a Semaphore class backed by monitors (i.e. the Lock and CondVar classes).
Neither of the methods should require more than five lines.
Assume **Mesa**-scheduled monitors.

```
public class Semaphore {
     Lock lock; // every monitor has a Lock and CondVar
     CondVar c;
     Int value;  // semaphores have a positive integer value

public Semaphore(int initialValue) {
     value = initialValue;
     lock = new Lock();
     c = new CondVar(lock);
}

public void P() {
/* YOUR CODE HERE */

}

public void V() {
/* YOUR CODE HERE */

}
```

How would the implementation change for **Hoare**-scheduled monitors?