**Problem 4**

1. Spinlocks are not appropriate for single-processor systems because they require busy waiting. Using a spinlock on a single-processor system would waste time and resources. Consider a system that utilizes time slice scheduling, where two processes A and B, both utilizing spinlocks, need to run. If a thread in process A acquires the critical section and the process uses up its time slice before finishing the critical section, then process B runs and gets to the spinlock to wait for the critical section, process B then needs to wait until it uses up its time slice so that process A may finish the critical section and set the condition for process B to break out of the spinlock. This situation has a lot of wasted cycles. In the worst case, the spinlock could cause a deadlock on the processor and there exist no other processors to break the deadlock. However, spinlocks are often used on multiprocessor systems because a concurrently running process could break the deadlock on another process by breaking a thread that is in a spinlock out of the lock loop.
2. Disabling interrupts is not sufficient for a multi-processor system because it does not guarantee mutual exclusion from multiple processes on multiple processors. This means that code running on other processors could access a resource that should be protected in a critical section even though the processor that is supposed to have exclusive access to the critical section is running that section. The processor that is first running the critical section can only provide mutual exclusion from other processes on the same processor.

On a single processor system, disabling interrupts could work because the processor can only run one process at a time. There are no other concurrent processes running on the system, so the process running the critical section will have mutually exclusive access to the critical section. On this kind of system, interrupts are the only way a process can lose control of a critical section.

On a multi-core system, we can make this approach work by imposing the requirements that you can choose which processor an interrupt goes to. When you implement a synchronization primitive, you must implement two functions: lock and unlock. In the unlock function, an interrupt should be sent to signal that the lock is ready to be unlocked before unlocking. A disadvantage of this strategy is that the wrong lock may be unlocked if the interrupt is sent to the wrong processor.