Homework 4 Synchronization & Multicore

1. Explain why spinlock is not suitable for single-processor system but used in many multiprocessor systems? Is there any solution?

2. In the course, we have learned the case of reader-friendly read-write locking. This kind of locking can cause writer to starve, why? Can you give a design of read-write locking which favors writer (avoids writer starvation)?

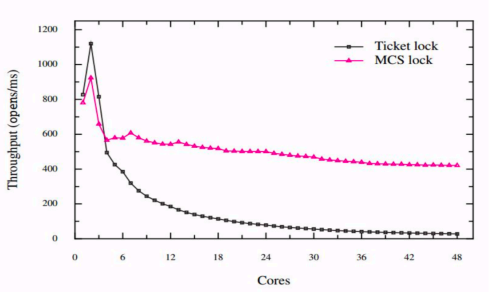
3. Assume there is a system with four processors and five identical resources. Each processor needs at most two resources within a limited time duration. Please show why this system is deadlock-free (all processors will eventually make progress).

4. ticket spinlocks can cause performance collapse under high contention. The figure below describes the performance of open system call using ticket locks and MCS locks. Please specify which code makes ticket lock's throughput drop. How does MCS lock achieve scalability? (The left figure is the code of ticket lock)

```
struct lock {
        volatile unsigned owner;
        volatile unsigned next;
}

void lock_init(struct lock *lock) {
        /* Initialize ticket lock */
        lock->owner = 0;
        lock->next = 0;
}

void lock(struct lock *lock) {
        volatile unsigned my_ticket =
                atomic_FAA(&lock->next, 1);
        while (lock->owner != my_ticket) {
                /* Busy looping */
        }
        barrier();
}
```



5. In the course, we have learned the NUMA-aware cohort lock. Although it can improve the performance in NUMA architecture, it introduces the problem of fairness, why? Can you give a method to solve this problem? (Recall what you learned from scheduling)

6. You are implementing a spinlock on the ARM CPU and you find that the critical section is not well protected by you locking. After you learn about the Weak-ordering Consistency, you know you should add barrier to your code. Please add barrier() in the proper place.

```
void lock(struct spinlock* lock) {

    /* locking */

}

void unlock(struct spinlock* lock) {

    /* unlocking */

}
```