HW 3 CS 3420

Problem 1

1. Meets Safety
    a. We must show that at each line, if a context switch occurs, we are still guaranteed that only one process will be in the critical section at a time. Regardless, when the CPU shifts from process 0 or process 1, intent[0] and intent[1] can both be true, but turn can either be 0 or 1. Hence, only one process can go into the critical sections at a time, while the other loops and waits.
2. Meets Progress
    a. There is no possibility of either livelocks ("you first, you first") or deadlocks ("me first, me first"). Note that if process 0 is in its CS, then intent[0] = true and turn = 0. Once it has completed it's CS, it must at some point set intent[0] = false. Because process 1 has not given up its intent to go into its CS and is busy looping checking and waiting, once P0 sets intent[0] = false, P1 is now free to go into its CS.
3. Meets Fairness
    a. Similarly to progress, if P1 is waiting, P0 must set intent[0] = false after completing its CS. As P1 is continuously waiting and loop checking if it's P1's turn, P1 will go into its CS at some point if intent[1] = true.

Problem 2

This approach is fair, as once the lock is released, whatever processor is able to decrement first will be the new owner of the lock as this is implemented as FIFO.

```
// globally shared variables
int simple_lock; // use as needed for correctness
int next_available; // next available ticket
int currently_serving; // ticket currently being served
// lock/unlock function prototypes
void lock(int *lock_ptr); // simple atomic t&s-based lock
void unlock(int *lock_ptr); // simple atomic unlock

void ticket_lock() {
      int my_ticket;
      lock(&simple_lock);
      my_ticket = next_available;   // grab a ticket
      unlock(&simple_lock);
      if (my_ticket == currently_serving) {
            return; //get served
      }
      else {
            lock(&simple_lock);
            currently_serving = next_available;
            next_available++;
            unlock(&simple_lock);

      }
}

void ticket_unlock() {
```

```
// yield to next ticket’s holder
      lock(&simple_lock);
      next_available++;
      unlock(&simple_lock);
}
```