Systems and Networks **Homework 3: Synchronization**
CS3600 Fall 2014 **July 28, 2014**

This homework is due at the beginning of class on October 8, 2014 and is worth 1.5% of your grade.

Name: ______________________________

CCIS Username: ______________________________

Problem	**Possible**	**Score**
1	25	
2	25	
3	25	
4	15	
5	10	
Total	100	

1. Suppose we slightly modified Peterson's solution to be the following

```
boolean flag[2] = {FALSE, FALSE};
int turn = 0;

1:  void almostPeterson(int i) {
2:      int j = 1-i;
3:      while (1) {
4:          flag[i] = TRUE;
5:          turn = i;                                      // modified
6:          while (flag[j] && (turn == j))) {}

7:          [critical section]

8:          flag[i] = FALSE;
9:      }
10:  }
```

Would this solution still be correct if two threads are executing `almostPeterson(0)` and `almostPeterson(1)`? If so, argue why it is the case. If not, give a counterexample that results in both threads being in the critical section at once. Your counterexample should be a sequence of instructions *x.y*, meaning process *x* executes line *y*. For example, the first few elements of the sequence may be {1.1, 1.2, 2.1, 1.3, ... }. (25 pts)

2. The following "solution" to the mutual exclusion problem for two processes was published in 1966 in the *Communications of the ACM*. Process 1 invokes `bogusMutex(0)` and process 2 invokes `bogusMutex(1)`. Only individual load and store instructions can be assumed to be atomic.

```
boolean blocked[2] = {FALSE, FALSE};
int turn = 0;

1:  void bogusMutex(int pid) {
2:      while (1) {
3:          blocked[pid] = TRUE;
4:          while (turn != pid) {
5:              while (blocked[1-pid]) {}
6:
7:              turn = pid;
8:          }

9:          [critical section]

10:         blocked[pid] = FALSE;
11:     }
12:  }
```

Find a counterexample that demonstrates that this solution is incorrect for two process on a single processor with preemptive scheduling. Show the exact sequence of events that leads to incorrect behavior. Your counterexample should be a sequence of instructions *x.y*, meaning process *x* executes line *y*. For example, the first few elements of the sequence may be {1.1, 1.2, 2.1, 1.3, ... }. (25 pts)

3. A dentist's office consists of a waiting room with n chairs and the examination room containing the dentist's chair. If there are no patients to be served, the dentist goes to sleep. If a patient enters the waiting room and all chairs are occupied, the patient leaves the office. If the dentist is busy, but chairs are available, then the patient sits in one of the free chairs. If the dentist is asleep, an arriving patient wakes up the dentist. The dentist serves patients in the order in which they took seats. When it is a patient's turn, (s)he vacates a seat in the waiting room, enters the examination room, gets a tooth extracted, then leaves the office. Write a program using either semaphores or a monitor to coordinate the dentist and the patients, which are each represented by a thread. (25 pts)

4. What is the worst possible *preemptive* scheduling algorithm, in terms of average waiting time? In other words, which *preemptive scheduling algorithm* maximizes average waiting time? You should assume that (a) you can never let the CPU be idle if there is a process that wants to run, (b) you know how many processes exist and exactly how many instructions each process will run before exiting, (c) context switches and your scheduling code take zero time, and (d) each process is entirely CPU-bound and will never relinquish the CPU willingly.

Note that you should consider all possible preemptive scheduling algorithms, not just ones that we talked about or (say) have a fixed time quantum. (15 pts)

5. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Prove that the system is deadlock-free. (10 pts)