**NAME:** _____ **(FIRST NAME FIRST)** **SCORE:** _____

# COSC 4330/6310 SECOND QUIZ JULY 8, 2015

*This exam is **closed book**. You can have **one page** of notes. UH expels cheaters.*

**1.** ***Questions with short answers:*** **(**6×5 points)

a) How do you pass a ***linked list*** to a ***remote procedure***? You send to the remote server an array containing the elements of the linked list preceded by a header containing the number of elements in the list and how to rebuild it.

b) What is the major disadvantage of ***non-preemptive scheduling*** policies? Non-preemptive scheduling policies let CPU-bound processes monopolize the CPU.

c) What is the difference between ***blocking sends*** and ***non-blocking sends***? A blocking send waits until the message is delivered while a non-blocking send does not.

d) What is the major disadvantage of the ***at-most-once*** semantics for remote procedure calls? The at-most-once semantics for remote procedure calls does not prevent partial executions of calls.

e) What is the main advantage of ***datagrams*** over ***streams***? They are faster because sending datagrams does not require establishing a connection between the sender and the receiver.

f) What is the sole proper way to initialize a ***mutex*** semaphore? Set it to ONE.

**2.** When does the System V Release 4 scheduler: (3×5 points)

**a)** ***Decrease*** the priority of a process? When a process has exhausted its CPU time slice.

**b)** ***Increase*** that priority? In two cases:

-- When a process returns from the wait state ("sleep" state)/when a process has performed a blocking system call.

-- When it has remained for too long in the ready queue without getting any CPU time.

**3.** Consider the following solution to the mutual exclusion problem and explain when it fails (5 points) and what happens then. (5 points)

```
int mark[2] = {0, 0}; // global variable

void enter_region(int pid) {  // 0 or 1
        mark[pid] = 1;
        while (mark[1 - pid]);
} // enter_region

void leave_region(int pid) {
        mark[pid] = 0;
} // leave_region
```

**When** two processes try to enter the critical region in lockstep.

**___ It denies mutual exclusion.**

**_X_ It denies access to the critical section.**

**4.** Indicate which of the following statements are ***true*** or ***false*** by circling the correct choice (5×3 points)

**T** **F** ***Monitor conditions*** can only have ***positive values***.

**T** **F** It is generally a good idea to assign ***fixed priorities*** to ***real-time processes.***

**T** **F** You can eliminate the ***hold and wait*** necessary condition for deadlocks by requiring all processes to acquire their resources in the same linear order.

**T** **F** Deadlocks can occur whenever ***any*** of the four Haberman's conditions is satisfied

**T** **F** **V( )** operations on semaphores are ***always non-blocking***.

**5.** When should you use ***spin locks***? (5 points) For short waits on multiprocessor architectures.

***Why***? (5 points) Short waiting times are likely to be shorter than the two context switches required for moving the waiting process to the wait state. In addition, a process doing a spin lock will not interfere with processes running on different processors.

**6.** A diner can seat a maximum of 40 customers. Complete the following code fragments to ensure (a) that the diner will never contain more than 40 customers and (b) that customers arriving together will be seated together. (*Hint: You will need a while loop somewhere.*) (20 points).

```
class Diner {
        private int nfree ;//diner is full when nfree === 0
        private condition notfull;

    public synchronized void arrive (int npatrons) {

            while (nfree < npatrons)

                notfull.wait;

            nfree –= npatrons;

    } // arrive

    public synchronized void leave(int npatrons) {

            nfree += npatrons;

            notfull.signal;

    } // leave

    diner(){
            nfree = 40 ;
    } //constructor
} // Diner
```