NAME: _____ (FIRST NAME FIRST) SCORE: _____

COSC 4330 SECOND MIDTERM APRIL 2, 2012

This exam is closed book. You can have one page of notes. Please answer every part of every question.

1. For each of the statements below, indicate in one sentence whether the statement is true or false (2 points), and ***why*** (3 points).

(a) Most servers use ***non-blocking receives***.

FALSE, they use blocking receives to avoid busy waits.

(b) Making all remote procedures ***idempotent*** greatly simplifies the task of the RPC server.

TRUE, the client can resend any request for which it did not get a reply.

(c) Peterson's algorithm for mutual exclusion does not work on ***multicore architectures***.

FALSE, it only assumes that instructions cannot be executed out of order.

(d) A ***blocking send*** blocks until the kernel accepts the message for delivery.

FALSE, it blocks until the receiver reads or retrive the message.

(e) A ***mutex semaphore*** can only have two correct values.

TRUE, these values are zero and one.

(f) ***Starvation*** is always the result of a ***deadlock***.

FALSE, starvation can also happen when one or more low priority processes cannot get access to the CPU.

2. Answer in ***one or two sentences*** to the following questions: (4×5 points)

(a) What is the difference between a **`signal`** and a **`notify`**?

A monitor procedure P doing a notify() does not risk to lose control of the monitor as procedures waiting on its notify will have to wait until procedure P terminates before gaining control of the monitor.

(b) What is the difference between a ***monitor condition*** and a ***semaphore*** ?

Monitor conditions have no value while semaphores do have one.

(c) What is the major disadvantage of the ***at-most-once*** semantics for remote procedure calls?

At-most-once semantics does not prevent nor detect partial executions of the remote procedure.

(d) What should a process do when it receives a ***retransmission*** of a message it has already received and acknowledged?

It should acknowledge that retransmission (in order to avoid additional retransmissions).

3. Alice and Barbara have to go to New Orleans to present a paper at a computer conference. They decide to meet at Hobby airport before catching their flight. Their main problem is that they can only communicate through ***blocking*** send and receive primitives with ***direct naming*** such as:

- **`bsend(Alice, buffer, nbytes)`**
- **`breceive(Alice, buffer, &nbytes)`**

How will they be able to wait for each other using these two primitives? (2×5 points)

Alice will do:	***Barbara will do:***
bsend(Barbara, buffer, nbytes)	breceive(Alice, buffer, &nbytes)

4. An interstate bus that can seat 44 passengers has a single door that lets passengers get in or out one by one at scheduled stops. Complete the two following methods to ensure that (a) the bus will never be overloaded and (b) passengers will not bump into each other when getting in or out the bus. (5×5 points)

```
Class bus {
      private int n_freeseats;  // number of free seats
      private condition not_full;

      public void synchronized board_bus() {

            if (n_freeseats == 0)

            not_full.wait;

            n_freeseats--;

      } // board_bus()

      public void synchronized leave_bus() {

            n_freeseats++;

            not_full.signal;

      } // leave_bus()

      bus() {
            n_freeseats = 44;
      } // constructor
   } // Class bus
```

5. What is wrong with the following solution to the mutual exclusion problem for ***two processes***?

```
int lock[2]; // lock is the only global variable
lock[0] = lock[1] = 0;  // 0 means FREE

enter_region(int pid)  { // pid must be 0 or 1
      lock[pid] = 1;  // 1 means BUSY
      while (lock[1-pid]);
} // enter_region

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

What happens? (5 points) The solution will cause a deadlock.

When does it happen? (10 points) When two processes try to enter the critical section in lockstep: each process will prevent the other from entering.