Name:

Operating Systems
Fall 2015
Test 2
November 9, 2015

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. You do not need to use a calculator.

You have 75 minutes to solve 6 problems, with an optional 7th problem for extra credit. As with any exam, you should read through the questions first and start with those that you are most comfortable with.

Partial credit will be offered only for meaningful progress towards solving the problems.

Please read and sign below if you agree with the following statement:

In recognition of and in the spirit of the academic code of honor, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:_____

1	/20
2	/10
3	/10
4	/10
5	/10
6	/15
Total	**/75**
7 (extra)	/15

1. [20 points] Short Answer Questions:

a. The term _____ refers to a technique in which a process can do nothing until it gets permission to enter its critical section but continues to execute an instruction or set of instructions that tests the appropriate variable to gain entrance.

b. *True or False:* Race condition is a situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work.

c. A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution is a _____ .

d. _____ is when the sequence of instruction is guaranteed to execute as a group, or not execute at all.

e. *True or False:* A process that is waiting for access to a critical section does not consume processor time when mutual exclusion is implemented with *CompareAndSwap*-based locks.

f. *True or False?* It is possible for one process to lock the mutex and for another process to unlock it.

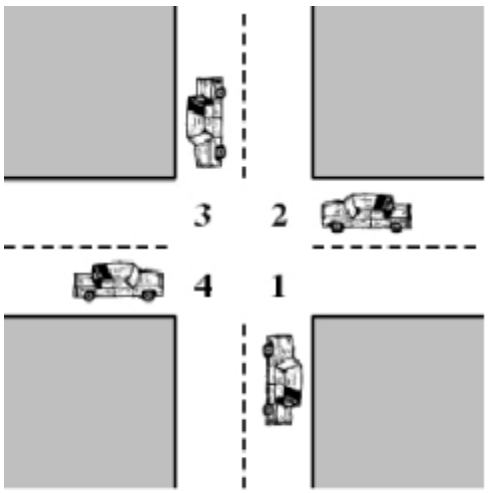
g. *True or False?* Deadlock avoidance is more restrictive than deadlock prevention.

h. *True or False?* A process that is waiting for access to a critical section protected by semaphores does not consume processor time.

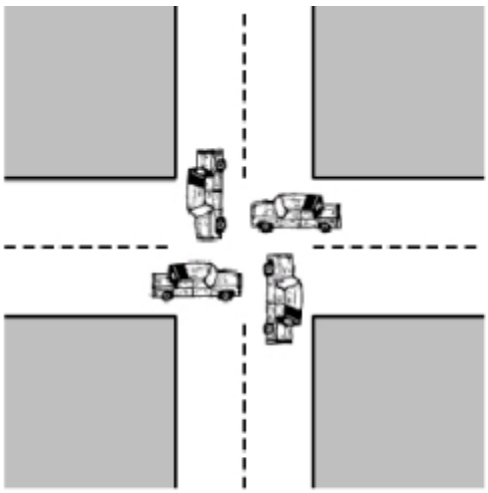
i. *True or False?* Multiple threads in the same process share all code and data resources.

j. *True or False?* Hardware support is always needed to synchronize more than two processes.

2. [10 points] Show that the four conditions of deadlock apply to the situation depicted below. Consider the intersection partitioned in 4 areas (or resources), numbered 1 to 4.



(a) Deadlock possible



(b) Deadlock

3. [10 points] Suppose we have the philosophers at a table as in the Dinning Philosophers problem, but the chopsticks are placed in a tray at the center of the table when not in use. As in the original problem, any philosopher can eat only with two chopsticks, but unlike in the original problem, any two available chopsticks will do.

   One way to prevent deadlock in this system is to provide sufficient resources. For a system with *n* philosophers, what is the *minimum* number of chopsticks that ensures deadlock freedom? Why?

4. [10 points] Process A is already written and requests the printer, the plotter, the tape drive, and the robotic arm, in that order. You need to develop processes B and C. Both need the printer and the arm, B needs the tape drive, and C needs the plotter. In what order should the requests be made? Why?

5. [10 points] Consider the following program executed by two different processes P1 and P2:

```
{
     shared int x;                          //s1
     x = 10;                                //s2
     while (1) {
          x = x - 1;                        //s3
          x = x + 1;                        //s4
          if (x != 10)                      //s5
               printf("x is %d",x)          //s6
     }
}
```

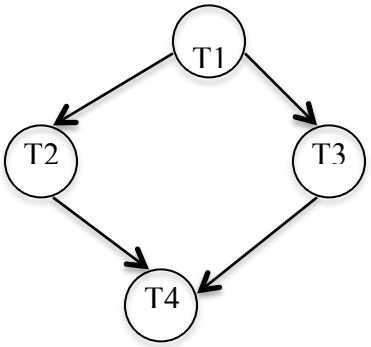
Consider that the processes P1 and P2 are executed on a uniprocessor system. Note that the scheduler in a uniprocessor system would implement pseudoparallel execution of these two concurrent processes by interleaving their instructions, without restriction on the order of the interleaving.

a. Show a sequence (i.e., trace the sequence of interleavings of statements) such that the statement "x is 10" is printed. Suggested format:
   Pi: <instruction> <relevant new value >
b. Show a sequence that leads to the statement "x is 8" be printed.

Hint: Remember that the increment/decrements at the source language level are not done atomically, e.g., the assembly language code below implements the single C increment instruction (x = x + 1).

```
LD R0,X /* load R0 from memory location x */
INCR R0 /* increment R0 */
STO R0,X /* store the incremented value back in X */
```

6. [15 points] Assume you are given a graph that represents the relationship between four threads (T1, T2, T3, T4). An arrow from one thread (Tx) to another (Ty) means that thread Tx must finish its computation before Ty starts. Assume that the threads can arrive in any order. Use semaphores to enforce this relationship specified by the graph. Be sure to show the initial values and the locations of the semaphore operations.



// Semaphores definitions and their initial values

T1(void){	T2(void){	T3(void){	T4(void){
\\T1 computation	\\T2 computation	\\T3 computation	\\T4 computation
}	}	}	}

7. **Extra credit [15 points]: This is a more difficult problem. Credit is given only for meaningful progress towards a correct solution.**

   Assume you are building a special operating system that requires executing the expression:

$$z = F3(F1(x),F2(F3(y)));$$

where x, y, and z are integers and F1, F2, F3 are functions.

The functions F1() and F3() must execute on thread T1 while the function F2 needs to execute on thread T2.

a. Your job is to write the code to force the expression to be evaluated regardless of the order they are run by the CPU scheduler. Note you will have to add code to all three functions below.

```
int z, x, y; //shared variables

//Declare shared variables and semaphore with initial values here.
```

`void ComputeZ() {`
 	
`StartThread(T1);`
 	
`StartThread(T2);`
 	
`return;`
`}`
	`void T1() {`
	
	`}`
		`void T2() {`
		
		`}`

b. Does your solution handle the following cases? If so, explain why. If not, explain the problem and how you could fix it.

i. Would your solution handle the case when ComputeZ() is called twice in a row like:

```
x=...;  y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z);
x=...;  y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z);
```

ii. Would your solution handle the case when ComputeZ() is called twice from two different threads? For example:

```
void Z1() {
x=...; y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z); }

void Z2() {
x=...; y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z); }

StartThread(Z1);
StartThread)(Z2);
```