Computing Science 300 **Quiz 2** **Sample problems**

1. Consider a process running on a LINUX system. Answer each of the following questions. The answer to each question inside each part of this problem should be no more than two sentences long.
   a) [6 points] The first thread in the process sets a global variable, myVariable, to have a value 3. The process then creates a second thread. Is the value of the global variable, myVariable, in the first thread the same as the value of the global variable, myVariable, in the second thread immediately after the second thread is created? Does the value of the global variable, myVariable, always have the same value in the second thread and in the first thread? Briefly Explain why?
   b) [6 points] The first thread in the process sets an automatic local variable, myVariable, to have a value 3. The process then creates a second thread. Is the value of myVariable in the first thread the same as the value of the variable myVariable, in the second thread immediately after the second thread is created? The second thread runs, sets myVariable to a value of 23 and then tells the first thread to run again. What is the value of the local myVariable, in the first thread when it begins to run for the second time? Why?
   c) [8 points] Consider scheduling of processes and user level threads. Is it possible for the threads in the process to be scheduled onto different processors in a multiprocessor CPU? Briefly explain why.

2. Consider the following program.

```
const int n=10;
int tally=0;
void total( ) {
    int count;
    for (count = 0; count < n; count++) {
            tally ++;
    }}
```

The main program using this function sets tally to 0, then starts two concurrent copies of process total. (Process total runs function total). Assume the two processes can execute at any relative speed and that a value can only be incremented after it has been loaded into a register by a separate machine instruction. (That is tally++ is executed in three machine instructions, load into register, then increment, then load back into variable in memory). You may assume that all other instructions (other than incrementing tally) are never interrupted between the load and increment or between the increment and load back steps

a) **[10 points]** Determine the upper bound on the final value of the shared variable tally after both copies of total( ) have run to completion. For the order of execution that produces the upper bound give a step by step time ordered description detailing the lines of code executed by each process each time it is in the CPU .

b) **[30 points]** Determine the order of execution through the pair of processes that produces the answer tally = 5 after both copies of the process total() have run to completion. For the order of execution that produces tally=5 give a step by step time ordered description detailing the lines of code executed by each process each time it is in the CPU .

3. **[40 points]** Consider the pseudo code below. The pseudo-code is given for parts of two processes P0 and P1 that will be sharing the CPU in a multiprocessing operating system. Mutual exclusion is being used to share a resource between these two processes. Assume that when the first block of code below to run begins to execute its loop that both P0_wants and P1_wants are zero.
   For each part of this problem give a time ordered instruction by instruction list of a particular path of execution through processes P0 and P1. Each of these paths of execution must describe
   - Which instructions each process runs when it begins a quanta in the CPU and which instructions it partially completes or completes during the quanta before it leaves the CPU.
   - The tasks completed by the instructions that are executed

   a) Give a path through the two processes for which the resource is successfully protected by mutual exclusion. The path of execution you give MUST include each process leaving the CPU at least twice.
   b) Give a path through the two processes for which the resource is NOT successfully protected by mutual exclusion because deadlock occurs. The path of execution you give MUST include each process leaving the CPU at least twice.

```
1) int P0_wants;
2) while(true) {
3)       P0_wants = true;
4)       while (P1_wants ) { };
5)       CriticalSection();
6)       PO_wants = false;
7)       NonCriticalSection();
8) }
```

```
1) int P1_wants;
2) while(TRUE) {
3)       P1_wants = TRUE;
4)       while (P0_wants ) { };
5)       CriticalSection();
6)       P1_wants = FALSE;
7)       NonCriticalSection();
8) }
```