CS 414 Assignment 3
10 points per answer for 1, 4a-c, 6a-b, 15 points per answer for 2a-b and 30 points each for questions 3, 5 (150 max). Due Wednesday February 22.

Short-answer questions (just a sentence or two each,please).

1. In class, we saw that an atomic **test_and_set** operation can be used to implement critical sections on a machine supporting parallelism (with multiple physical CPUs). But suppose that you were given a different atomic instruction called **decrement**. This instruction decrements a variable and leaves the initial value in register 0. Could **decrement** be used to implement critical sections? Show us how or explain why not.

2. Consider the N-process Bakery Algorithm:

```
CSEnter(i):
        chosing[i] = true;
        number[i] = max(number[0], ..., number[N-1])+1;
        chosing[i] = false;
        for(j = 0; j < N; j++) {
                while(chosing[j]) continue;
                while(number[j] > 0 && (number[j],j) < (number[i],i)) continue;
        }

CSExit(i):
        number[i] = 0;
```

   (a) Suppose that process 2 is trying to enter the critical section and has already picked a number value and set chosing[2] to false.. In a "worst case scenario", how many times could process 0 "sneak in" before process 2 gets its turn? Explain.

   (b) Same question, but now answer for the case where process 2 hasn't yet set chosing[2] to false.

3. In the Bakery Algorithm, it may seem that some of the variables used are redundant. Suppose the Bakery Algorithm is modified so that references to the Boolean array chosen is commented, and the new algorithm is:

```
CSEnter(i):
        /* chosing[i] = true; */
        number[i] = max(number[0], ..., number[N-1])+1;
        /* chosing[i] = false; */
        for(j = 0; j < N; j++) {
                /* while(chosing[j]) continue; */
                while(number[j] > 0 && (number[j],j) < (number[i],i)) continue;
        }

CSExit(i):
        number[i] = 0;
```

   Will this modified algorithm effectively solve the critical section problem? If yes, prove it; else give a case in which this algorithm will fail!!

4. In class, we developed a proof that the Bakery Algorithm is both safe (mutual exclusion is preserved) and live (the algorithm guarantees progress and bounded waiting). See section 6.2 of the textbook for definitions. But our reasoning depended on the assumption that any process that enters its critical section will exit from the critical section within finite time. Suppose that a process were to enter its critical section but crashes while in it - it calls CSEnter() but does not call CSExit().

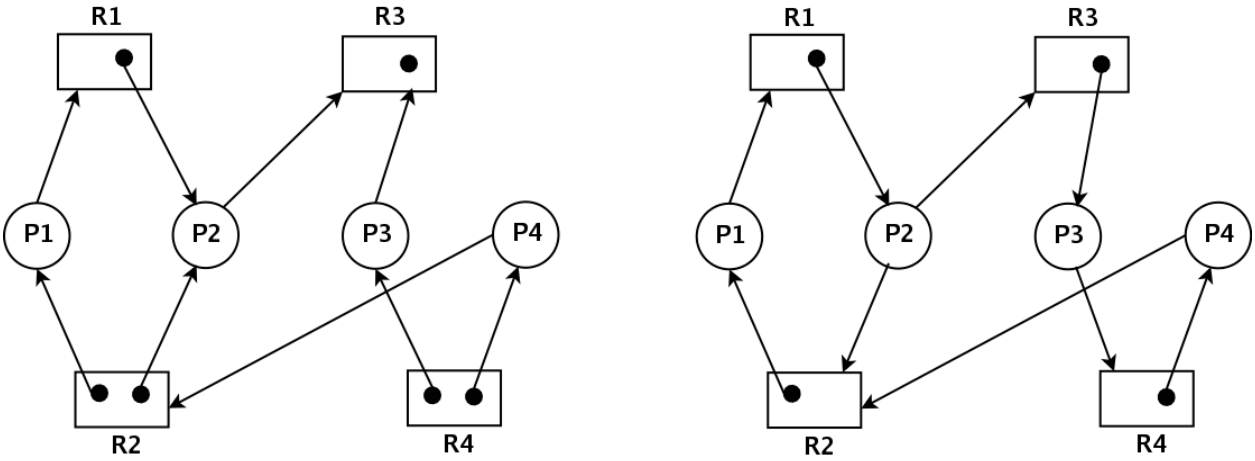

Figure 1: 6 (a) , (b)

(a) Could the safety property of the solution be violated? Explain very briefly.

(b) Could the progress part of the liveness property be violated? Explain very briefly.

(c) Could the bounded waiting part of liveness be violated? Explain very briefly.

5. The robots in an automotive assembly line are of two types - welding units and bolt-tightening units. There are two welding units and three bolt tightening units. The two welding units can be active at the same time, doing welds on the left and right side of a vehicle, or the three bolt-tightening units, but not both types of units at the same time. Each unit fetches the raw material (welding rod/bolt) and performs its action. The life of the robot is:

   Loop:
   load raw materials
   perform action (weld or tighten bolts)
   goto loop

   Modeling each robot as a concurrent process in a shared memory space, write a *monitor* implementing the necessary synchronization. Also provide a skeleton for the *welder* process and the $bolt - tightener$ process, so that we can see exactly how the monitor entry points are called.

6. In the resource allocation graphs given in figure 1 either show a reduction order that proves that the graph does not represent a deadlock or give an irreducible component and hence establish that a deadlock is present.