T-106.5600 Concurrent Programming Examination August 2011/ hsa

Be concise and clear. The shorter you can be, the better.

1. Consider the simple bakery algorithm for two-process critical section:

Algorithm 5.1: Bakery algorithm (two processes)

integer np ← 0, nq ← 0

p		q	
	loop forever		loop forever
p1:	non-critical section	q1:	non-critical section
p2:	np ← nq + 1	q2:	nq ← np + 1
p3:	await nq = 0 or np ≤ nq	q3:	await np = 0 or nq < np
p4:	critical section	q4:	critical section
p5:	np ← 0	q5:	nq ← 0

a) Show the safety with a proper invariant.

b) Which of the statements in 5.1 contain more than one critical reference, i.e. they are not trivially atomic in the classical LOAD/STORE atomic register model?

c) Find a scenario where the mutual exclusion does not hold when the statement p2 (symmetrically for q2) is replaced by two statements "p2.1: temp <- nq" and "p2.2: np <- temp +1".

2. Show that Algorithm 5.1 as modified in the previous question is safe, if the statement "p1.2: np <- 1" is added before p2.1, and symmetrically for q.

3. Solve the five dining philosophers problem using tuple-space, so that the utilization rate of the forks is maximal, i.e. if a philosopher could eat, he will get to eating immediately. Hint: If a philosopher can not get to eating, he is put into an explicit intermediate state "hungry" so that when his neighbour stops eating, he can awakened. Define clearly the meaning of the different tuple types used and attach appropriate tags to them. The get_to_eat() and end_eat() operations should be simple algorithms using no other global variables except tuples.

Linda tuple-space primitives are: `postnote ('tag', ...)`, `readnote ('tag', ...)`, `removenote ('tag',...)`. Indicate clearly in a `readnote(...)`, or `removenote(...)` operation when a matching tuple with an element value equal to a program variable value is sought for (syntax " v="), from the case where an element value of a otherwise matching tuple is just assigned to a program variable (syntax " v").

4. One-lane bridge. Cars coming form north and south have to cross a river along a very long and narrow one-lane bridge. Cars driving to the same direction may be on the bridge at the same time, but cars heading to opposite directions can't. Consider the following generic monitor solution `One_lane_bridge` to the problem, where the cars are processes calling the public methods `cross_from_North()` and `cross_from_South()`.

```
     monitor One_lane_bridge {
     int ns =0;            //north-south cars on the bridge
     int sn =0;            //south-north cars on the brigde
     cond ns_c:            //condition to enter from north
     cond sn_c;            //condition to enter from south
     private procedure startNorth() {
           if (sn > 0) wait(ns_c);
           ns++
           }
     private procedure endSouth() {
           ns--;
           if (ns == 0) signal_all(sn_c));    //signals possible waiting sn cars
           }
   public cross_from_North() {   // this is needed to provide a simpler API
     startNorth();
     // north-south crossing operation is embedded here
     endSouth();
          }
```

```
        // the south-north direction is symmetric
```

Show by using proper invariants that the solution is a) safe and b) does not cause any unnecessary waiting.

5. a) Transform the example code in question 4 to a similar Java version.
   b) Refine the Java version to be fair, i.e. so that cars from each direction will pass the bridge in finite time.