T-106.5600 Concurrent Programming Examination March 2012/ hsa

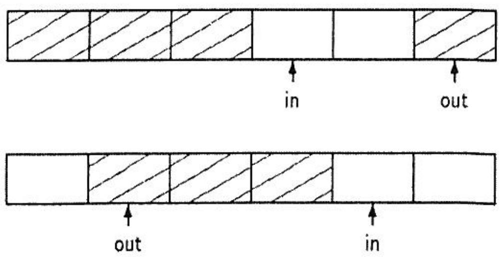
Be concise and clear. The shorter you can be the better. The simplicity of the solutions and answers is an important grading criterion! NOTE: There are only 4 questions in this exam!

1. Prove with proper invariants that the following simple two-process version of the bakery algorithm for the critical section is safe.

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

Does it satisfy the necessary liveliness requirements? Prove this with temporal logic or disprove with a contradicting example.
Analyze the atomicity of its statements. Are they minimal to achieve safety, or could they be broken to smaller ones?

2. A bounded buffer is frequently implemented as a circular buffer, which is an array indexed modulo its length. The index variable in points to the next empty space for a producer and out the next full for a consumer (see the picture below on the left). Prove with proper invariants that the following algorithm, where synchronization between a producer and a consumer is solved with semaphores works correctly.



Algorithm 6.19: Producer-consumer (circular buffer)			
dataType array [0..N] buffer integer in, out ← 0 semaphore notEmpty ← (0, ∅) semaphore notFull ← (N, ∅)			
	producer		consumer
	dataType d		dataType d
	loop forever		loop forever
p1:	d ← produce	q1:	wait(notEmpty)
p2:	wait(notFull)	q2:	d ← buffer[out]
p3:	buffer[in] ← d	q3:	out ← (out+1) modulo N
p4:	in ← (in+1) modulo N	q4:	signal(notFull)
p5:	signal(notEmpty)	q5:	consume(d)

Analyze the situation if several producers and consumers access the buffer at the same time. Is it still correct or should it be enhanced and how?

3. One-lane bridge. Cars coming form north and south have to cross a very long and narrow one-lane bridge. Cars driving to the same direction maybe on the bridge at the same time, but cars heading to opposite directions are not allowed. Consider the following Java code outline for the solution to the problem, where the cars are threads calling the public methods `cross_from_North()` and `cross_from_South()` of the class `One_lane_bridge`. Complete the missing peaces of the synchronization logic using Java. Do not worry about the performance or fairness first, however the directions should be treated symmetrically. Be clear when defining the needed variables! Write down two invariants specifying:
1) There can be several cars on the bridge at the same time, but they must all head to the same direction.
2) There is no unnessary waiting. Give an argument that both of them they are preserved.

```
class One_lane_bridge {
      ...
      private synchronized void startNorth() {
         ...
      }
      private synchronized void endSouth() {
       ...
      }
      public void cross_from_North() {
         startNorth();
         // north-south crossing operation is embedded here
         endSouth();
      }
      ...
         // south_north crossing is symmetrical
      }
   }
```

Analyze the performance of your solution and suggest potential improvements to it.
Analyze the fairness of your solution and suggest potential improvements to it.

4. Tuple Space. Write a tuple-space solution to the one-lane bridge problem. Define clearly the meaning of the different tuple types used and attach appropriate tags to them. No global variables except tuples should be used. The simpler your solution is, the better. Discuss the pros and cons of your solution. Use tuple-space operations: `postnote ('tag', ...)`, `readnote ('tag', ...)`, `removenote ('tag',...)`. Indicate clearly in a `readnote(...)`, or `removenote(...)` operation when a matching tuple containing element equal to the value of a program variable v value is sought for with syntax " v=" from the case where an element value of an otherwise matching tuple is just assigned to a program variable (syntax " v").

Analyze the fairness of your solution and suggest potential improvements to it.