KTH Computer Science
and Communication

Hand-in 1
DD2451 + FDD3008, Parallel and Distributed Computing
Fall 2011

Posted Nov 4 16.00. Due Nov 11 16.00. Answers can be mailed to mfd@kth.se as pdf or dropped in Mads' intray, located on level 4, Lindstedtsvägen 3 (enter through the doors with the label "NADA", after going a small distance to the left proceed straight and you find the department mail boxes on your right). Make sure answers are clearly marked with name and mail account on each sheet. The general rules on homework solutions available at the course home-page apply. In particular, discussions of ideas in groups of up to at most two people are allowed but solutions should be written down individually, and you should note the name of your discussion partner. Some of the problems below are "classical" and hence their solutions are probably posted on the Internet. It is not allowed to use such solutions in any way. The order of the problems is "random" and hence do not expect that the lowest numbered problems are the easiest. Any corrections or clarifications on this problem set will be posted under "Exercises and hand-ins" on the course home page (google DD2451).

1. (40p) Consider the following mutual exclusion protocol:

```
class MaybeMutex implements Lock {
  boolean[] flag ;
  public MaybeMutex (int n) {
    flag = new boolean[n];
    for (int i = 0; i < n; i++) {flag[i] = false}
  }
  public void lock() {
    int i = ThreadID.get();
    label1: flag[i] = false ;
    for (int j = 0; j < i; j++) {
      if flag[j] = true {continue label1} };
    flag[i] = true;
    for (int j = 0; j < i; j++) {
      if flag[j] = true {continue label1} };
    label2: for (int j = i + 1; j < n; j++) {
      if flag[j] = true {continue label2} };
  }
  public void unlock() {
    int i = ThreadID.get();
    flag[i] = false;
  }
```

1a (10p) Prove, using interval assertions in the style of Herlihy and Shavit chapter 2, that MaybeMutex satisfies mutual exclusion.

1b (10p) Is MaybeMutex deadlock free? Is it starvation free? Either prove that deadlock freedom, alt. starvation freedom, holds or explain how a counterexample execution can be constructed.

1c (10p) Suppose the second for-loop is removed. Does the algorithm still satisfy mutual exclusion? Either prove that it does, or exhibit a counterexample.

1d (10p) Replace the boolean flag array in the MaybeMutex algorithm of exercise 1 by a safe boolean register array. Does mutual exclusion still hold? Again give a proof or produce a counterexample.

2. (30p) Read exercise 32 in H&S. The operations `getAndIncrement`, `get`, and `getAndSet` are atomic operations which in an uninterruptable fashion, resp.,
   a. Reads an atomicInteger register, increments it, and returns the original value of the register
   b. Reads the register and returns the value
   c. Reads the register, sets it to the argument value, and returns the original value of the register.

2a (10p) Give an execution showing that the linearization point for `enq()` cannot occur at line 15

2b (10p) Give an execution showing that the linearization point for `enq()` cannot occur at line 16

2c (10p) For one of the executions in 2a or 2b show in detail why it is linearizable. Use the definitions.

2d (Harder + optional: 20 bonus points for everyone) Sketch a proof that the HW queue is linearizable.

3. (10p) Prove that sequential consistency as determined by condition **L1** of def. 3.6.1 of H&S is non-blocking. Explain carefully each step in the proof.

4. (20p) Let *r* be an SRSW read write register. In each case below, give a careful argument or exhibit a counterexample:

4a (5p) *r* quiescently-consistent if and only if *r* safe

4b (5p) *r* is quiescently-consistent if and only if *r* is regular

4c (5p) *r* is sequentially consistent if and only if *r* is safe

4d (5p) *r* is sequentially consistent if and only if *r* is regular

Good luck!