UNIVERSITY OF OSLO

Faculty of mathematics and natural sciences

Examination in	INF2140 — Parallel Programming
Day of examination:	15. June 2012
Examination hours:	9.00 – 13.00
This problem set consists of 6 pages.	
Appendices:	None
Permitted aids:	None, i.e., no special exam resources are allowed.

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

Some general advises and remarks:

- This problem set consists of several independent parts. It is wise to make good use of your time.
- You can score a total of 100 points on this exam. The number of points stated on each part indicates the weight of that part.
- You can make your own clarifications if you find the examination text ambiguous or imprecise. Such clarifications must be written clearly in the delivered answer.
- Make short and clear explanations!

Good luck!

(Continued on page 2.)

Problem 1 Actions (weight 20)

1a Actions in FSP (weight 8)

Consider the following FSP processes:

```
A = (a -> b -> A  |   d -> A).
B = (b -> a -> B  |   d -> B).
C = (c -> C  |   d -> C).
||AB  = (A || B).
||ABC = (A || B || C).
```

1. Draw the state machine defined by process A.
2. Explain the behavior of ||ABC.
3. Is there a possibility of deadlock for ||AB?
4. Are there any progress problems for ||AB for some of the actions? Explain briefly.
5. Redefine A by hiding the action a. What is the resulting alphabet of A ? Is there now a possibility of deadlock for ||AB? Are there now any progress problems for ||AB with the redefined A?
6. Redefine C by extending the alphabet of C by $\{a, b\}$. Now consider A || B || C. Are there any deadlock or progress violations.

1b Shared actions in Java (weight 7)

How would you implement the action d of the model `||ABC` in a Java solution?

1c Buffers in FSP (weight 5)

Consider a system in FSP with connects a producer `PROD` to a consumer `CONS` by means of a buffer.

```
PROD = (in[i:0..3]-> PROD).
CONS = (out[i:0..3] -> CONS).
```

1. Define a *single-cell* buffer `BUF` in FSP; i.e., the buffer can at most contain one element and construct a system `SYS` which connects the buffer to the producer and the consumer.

(Continued on page 3.)

2. Define a *two-cell* buffer `BUF2` by composing two copies of `BUF` and construct a system `SYS2` which connects the buffer to the producer and the consumer.

3. Define `BUF3` by modifying `BUF2` to allow *two* different producers and construct a system `SYS3` which connects the buffer to the producers and the consumer.

Problem 2 Semaphores (weight 25)

2a Semaphores in FSP (weight 3)

Make an FSP process which models a semaphore.

2b Semaphores in Java (weight 3)

Semaphores can be implemented in Java as passive objects that react to up and down actions. Define a class in Java which implements general semaphores.

2c Job processing using semaphores (weight 7)

Consider a `Server` object which processes jobs from `Client` objects. Jobs have a given size. When a client wants to process a job, the client calls the method `request(size)` on the server (the parameter `size` is needed for Problem 2d). When this method returns, the client can process the job. When the jobs has finished, the client calls `release()` on the Server to allow another job to be processed. A `Server` class which processes jobs may be implemented as follows:

```
class Server {
  private bool free;

  public Server(){
    free = true; }

  public void request(Int size){
   free = false;
  }

  public void release(){
    free = true;
  }
}
```

Modify the `Server` class to make sure that jobs are processed *one at the time* and in the *order of arrival*. You should only use instances of the `Semaphore` class

(Continued on page 4.)

to synchronize the processes. In your solution, `request` and `release` should not be synchronized methods.

2d Shortest job first using semaphores (weight 12)

Assume that jobs may have the sizes 1, 2, 3,...,`maxsize` (where `maxsize` is a parameter of `Server`). Modify your server such that jobs are scheduled using a *shortest job first* strategy; i.e., if there are waiting jobs of different sizes, a job with a smaller size should be selected before a job with a larger size.

Problem 3 A Shared Shower Room (weight 55)

Imagine that there is one shower room to be used by both girls and boys, but not at the same time. Thus at any time, there can either be boys or girls in the shower, but not both. The shower room has a limited capacity of M, thus the number of persons in the shower room at a given time should be at most M.

Each boy or girl is supposed to repeat the following cycle of actions:

> *enter*, *shower*, and *leave*.

(possibly labeled or indexed). No other actions should be needed in the first subproblem below.

3a Process Modeling (weight 15)

Program the shower system in FSP by means of a monitor SHOWER and with a number N of BOY processes and a number N of GIRL processes ($N > M$). Define first processes BOY, GIRL, SHOWER, and afterwards the whole shower system SYS.

3b Deadlock (weight 2)

Explain briefly if there is any deadlock in the system SYS or not.

3c Safety (weight 8)

1. Define a safety property SAFE in FSP ensuring that a boy and a girl will never be in the shower at the same time.
2. Show how this can be checked in FSP for the shower system SYS.

(Continued on page 5.)

3. Will the safety property be satisfied for SYS?
4. Will the safety property be satisfied for the monitor process SHOWER alone?

Explain briefly.

3d Progress (weight 3)

Define a progress property in FSP expressing that some girl will eventually be able to enter the shower.

3e Adverse conditions (weight 3)

It is desired that a girl wishing to shower eventually will be able to do so even in a worst case scenario where boys continuously enter and use the shower whenever possible.

1. Use priorities to be able to check this fairness problem in FSP by redefining SYS.
2. Then define the desired progress property in FSP and discuss whether it is satisfied.

3f Fair System (weight 8)

1. Improve the shower system such that it is fair with respect to both girls and boys wishing to enter the shower. You may introduce new actions.
2. How can you show with FSP that the revised system is fair?

3g Java implementation (weight 10)

Make a Java implementation of the first version of the (unfair) monitor SHOWER. You do not need to consider fairness to boys and girls.

3h Java implementation: Notification (weight 3)

Explain whether you should use *notify* or *notifyAll* in your Java solution.

(Continued on page 6.)

3i Java implementation: Synchronization (weight 3)

Explain if all methods in the Java implementation need to by synchronized.