Concurrence, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
January 14th 2013

Answer the questions concisely and precisely
Answer each problem in a separate page (remember to put your name)
Closed-book exam
Duration: 1 hour

Exercise 1 FSP + JAVA (4 Points)
Let us consider a bank dealing with micro accounts. In a micro account you can just deposit or withdraw **one** euro in each operation. Moreover there is a bound M to the maximum quantity of money in the account.

1. Complete the following FSP expression. The alphabet of `MICRO_ACCOUNT` is `deposit` and `withdraw`. Take into account that it is not possible to withdray one euro if there is no money in the account and there is no possible to deposit one euro when there are M euros in the account.

```
const M=3

MICRO_ACCOUNT = MONEY[0],
MONEY[i:0..M]=...
```

2. Given the following (generic) client

```
CLIENT=(withdrawn->CLIENT|deposit ->CLIENT).
```

complete the following FSP expression `||BANK`. This expression has to describe two clients `alice` and `bob` sharing a common `MICRO_ACCOUNT`.

```
||BANK=.....
```

3. Write a monitor for `MICRO_ACCOUNT`. Please complete the following schema:

```
public class MicroAccount {
   private int i=0;
   private int M;

   public MicroAccount(int bound) {
     M=bound;
   }
...
}
```

4. Consider the following JAVA implementations of the CLIENT and ||BANK

```
public class  Client extends Thread{
   MicroAccount account;
   String name;

   public Client(String name, MicroAccount account){
      this.name=name;
      this.account=account;
   }

   public void run(){
      while(true){
         try{
            if(Math.random()<0.5) {
               account.deposit();
               System.out.println(name + " deposit one euro");
             } else {
               account.withdraw();
               System.out.println(name + " withdraw one euro");
             }
            Thread.sleep(3000);
             }catch (InterruptedException e){}
      }
   }
}
```

and

```
public class Bank {

   public static void main(String[] args){
      MicroAccount account= new MicroAccount(3);
      Client alice = new Client("Alice", account);
      Client bob = new Client("Bob", account);
      alice.start();
      bob.start();
   }
}
```

Taking into account the implementation of `MicroAccount` reason about possible deadlocks.

Exercise 2 Erlang (3.5 Points)
Create the Erlang function to be executed in a simple web server with the following functionalities:

- Handle connections: on receiving connections (tuple {`connect,IP`}), the server must create a web process (executing function `navigate` from module `web`) passing as parameter the `IP` of the connecting machine. A message `start` must be sent to the new process created to permit the browsing for machine `IP` throught the newly created web process. You dont have to provide the functionality for the web process, simply are required to create it and send the start message.
- Handle disconnections: on receiving the tuple {`disconnect,IP`}, the server must send an `abort` message to the web process devoted for the corresponding `IP`. If no web process exists for such `IP`, an error must be emited.
- Information: report the list of current connections when the message `info` is received.

Exercise 3 Lock-Free (2.5 Points)
Answer concisely the following questions:

1. What is a linearization point ?
2. What is hand-over-hand locking in the linked-list structures ?
3. When is better a lock-free implementation than a coarse-grain ?