Exam INF 223

SOLUTIONS

9 February 2011

1h30 – No Documentation Allowed (the marking scheme is indicative)

1 Concurrency

1.1 Threads Posix (2 points)

Provide an implementation of the *post* and *wait* operations on a semaphore using POSIX mutexes and condition variables. Comment your code.

1.2 Threads Java (4 points)

Five threads: T0, T1, T2, T3 and T4, are sharing a variable – `worker`, indicating which of them is allowed to work at any one time. When a thread wishes to work, it calls the `waitforMyTurn`() method. Only the thread whose identity (`this.worker`) matches `worker` is allowed to execute.

```
public synchronized void waitforMyTurn() {
while (worker != this.worker) {
      try {
      wait() ;
} catch (InterruptedException e) { }
}

// work
this.work() ;

// Tell the next thread it may work
worker = (worker + 1) % 5;
notify();
}
```

Assume the following scenario:

- `worker` is set to 3;
- The threads T1, T2 and T4 are blocked in the `wait()`method;
- T3 is currently in its `work()` method.

1.2.1 Question 1

This situation may lead to a deadlock. Describe an execution scenario that will lead the application threads into such a deadlock situation.

1.2.2 Question 2

Suggest a simple modification for providing a correct solution (that does not lead to a deadlock). Justify your answer.

Solution ----------

1. Question 1 : When thread T3 exits the work() method, it sets `worker` to 4 before calling notify(). One of the waiting threads is arbitrarly released. We supposed T2 is the one. T2 exits the wait method, checks `worker,` finds out that this is not its turn, so it enters wait() again.

   T0 and T3 will call `waitforMyTurn`() and will also stay in the wait() method, since `worker` is set to 4. Now, all threads are blocked in the wait method.

2. Question 2 : Use notifyAll() instead of notify()

End solution ----------

2 Communications

2.1 Question 1 : Sockets (2 points)

Explain accurately what is happening while executing the following instructions : what is the usage of Port, how will be released the server waiting on `accept` , what will it be doing once released ?

```
ServerSocket sock1 = new ServerSocket(Port);

Socket sock2 = sock1.accept();
```

3 Design Patterns

3.1 Comparison between CORBA and RMI (3 points)

3.1.1 Question 1

Compare the support of CORBA and RMI for the *heterogeneity*, *interoperability* and *portability* of distributed applications, by addressing the following questions:

3.1.2 Question 2

Define the concepts of heterogeneity, interoperability and portability, in the distributed systems context.

3.1.3 Question 3

Indicate the support for heterogeneity, interoperability and portability in distributed applications that use CORBA and RMI.

3.2 The Adapter Design Pattern (3 points)

3.2.1 Question 1

Describe the Adapter design pattern, by briefly indicating its application context, addressed problem and provided solution.

3.2.2 Question 2

Depict the static structure of the Adapter design pattern - e.g. via an UML class diagram.

3.2.3 Question 3

Describe the runtime behaviour of the Adapter design pattern - e.g. via an UML sequence diagram

4 Use Case (Case Study)

4.1 Question 1 : Thread version (2 points)

In the Thread version of the case study, explain the way in which the system, while in the `lock()` method, can be brought to send a message to another node. Explain the reason for which the reply to this message may come from a third node. (Justify your answer)

4.2 Question 2 : Socket Version (2 points)

In the Socket version of the case study, explain the way in which the nodes manage to get in contact with each other, while having no initial information on one other. (Justify your answer)

4.3 Question 3 : CORBA Version (2 points)

In the CORBA version of the case study, the `Peer` interface defines the `send()` method, as indicated below. How can this method be rendered asynchronous? Would such modification impact on the correct functioning of the case study? (Justify your answer)

```
interface Peer {
   void send (in long kind, in long from);
};
```