DUBLIN INSTITUTE OF TECHNOLOGY

DT211 BSc. (Honours) Degree in Computing

Year 4

DT228 BSc. (Honours) Degree in Computer Science

Year 4

WINTER EXAMINATIONS 2014/2015

Distributed Systems [CMPU4021]

[DT211/4, DT228/4]

Ms. E. Hatunic-Webster
Dr. D. Lillis
Mr. T. Nolan – DT211
Mr. K. Foley – DT228

.

Tuesday 16 Dec. 2014 9.30 a.m. – 11:30 a.m.

Attempt **3 questions**
All questions carry **equal** marks
One complimentary mark is available

1. (a) Discuss the difference between *peer-to-peer* and *client-server* software architectures and give examples of systems they are suitable for.

(8 marks)

(b) Compare *distributed* object systems and *non-distributed* object systems regarding:

- Method invocation
- Parameter passing
- Memory allocation / deallocation

(12 marks)

(c) Using *sample* code in Java, show how to create a UDP *server* that will allow guessing a number.
When the server is started it stores a random number between 1 and 50. The client reads numbers in from the user and sends these to the server. The server responds with a string saying HIGHER, LOWER to EQUAL. The client can continue entering values until it gets EQUAL returned.

(13 marks)

2. (a) Describe the *security requirements* for a typical on-line application running in a distributed system environment.

(8 marks)

(b) Consider the system where a *football-score-service* can provide scores to client applications in real time, as scores change.

Present a design for a distributed object system for such a service and illustrate the role of remote objects and proxy objects. Evaluate the strengths and weaknesses of your design.

(12 marks)

(c) Discuss *distributed event based* systems. Your answer should explain the roles of the participating objects.

(13 marks)

3. (a) Explain what is means for a Java thread to be:

- Runnable
- Blocked
- Waiting
- Deadlocked

(8 marks)

(b) Consider the four Java classes shown below:

```
public class Main {
      public static void main(String args[]) {
            MyData data = new MyData();
            new Thread(new Producer(data)).start();
            new Thread(new Consumer(data)).start();
            new Thread(new Consumer(data)).start();
      }
}
```

```
public class Consumer implements Runnable {
      MyData data;
      public Consumer(MyData data) { this.data = data; }
      public void run() {
            for (;;)
                  System.out.println ("Consumer: " + data.load());
      }
}
```

```
public class Producer implements Runnable {
      MyData data;
      public Producer(MyData data) { this.data = data; }
      public void run() {
            for (int i = 0; ; i++) {
                  data.store(i);
                  System.out.println ("Producer: " + i);
            }
      }
}
```

```
class MyData {
      private int data;
      private boolean ready;
      private boolean taken;
      public MyData() {
            ready = false;
            taken = true;
      }
      public synchronized void store(int data) {
            while (!taken);
            this.data = data;
            taken = false;
            ready = true;
      }
      public synchronized int load() {
            while (!ready);
            ready = false;
            taken = true;
            return this.data;
      }
}
```

Describe the most significant *synchronisation* problem that could occur while the program above is running. Show how to modify the code to prevent this problem from arising and explain why this prevents the problem from occurring.

(12 marks)

(c) Demonstrate how *two-phase locking* can be used to ensure *serial equivalence*. Show also how this can be extended to prevent problems such as *dirty reads* and *premature writes*.

(13 marks)

4. (a) Discuss the importance of *interfaces* in distributed object systems and show how interfaces are implemented in Java Remote Method Invocation (RMI).

(8 marks)

(b) Using diagrams and sample code, show how you would design and implement in *Java RMI* a *chat* application. You may use *pseudocode* for the detail, but you must precisely identify the required *classes* and *interfaces*. The application requirements are:

Clients connect to a centralised server. Clients can start a new chat room by supplying the theme for the room. Other clients can then lookup rooms by their theme. Once clients have either created a room or looked up a room, they can then send a message into the room.

(12 marks)

(c) With the aid of a diagram describe the Java Remote Method Invocation (RMI) *architecture*.

(13 marks)