CAVENDISH CAMPUS

School of Electronics and Computer Science

Modular Undergraduate Programme
Second Semester 2013 – 2014

Module Code: ECSE603

Module Title: Concurrent Programming

Date: Tuesday, 7th May 2014

Time: 10:00 – 12:00

Instructions to Candidates:

Answer THREE questions.
Each question is worth 33 marks.

Question 1

(a) Describe the Finite State Process (FSP) *abstract model* of a process. **[4 marks]**

(b) The following FSP process models a 4 minute egg timer.

```
EGG_TIMER( N = 240 ) = COUNTDOWN[ N ] ,

COUNTDOWN[ i : 0..N ]
    = (  when( i > 0 )   tick -> COUNTDOWN[ i - 1 ]
       | when( i == 0 )  soundBuzzer -> STOP
       | reset -> COUNTDOWN[ N ]
      ) .
```

(i) Explain the meaning of the following FSP language features used in the `EGG_TIMER` process: "`->`", "`|`", "`when ( i > 0 )`" and "`STOP`". **[12 marks]**

(ii) Explain the meaning of process *Alphabet*. Give the alphabet for the `EGG_TIMER` process. **[2 marks]**

(iii) Explain what *deadlock* means for an FSP process. Describe the `EGG_TIMER` process's deadlock trace? **[5 marks]**

(c) For each of the following FSP processes give the corresponding *Labelled Transition System (LTS) Graph*.

(i)
```
OYSTER = ( topUpOyster -> twentyPounds
           -> toppedUp -> printReceipt -> OYSTER ) .
```
[4 marks]

(ii)
```
TICKET
  = (  zone12 -> fourPounds -> printTicket -> TICKET
     | allZones -> tenPounds -> printTicket -> TICKET
    ) .
```
[6 marks]

Question 2

The following is a specification of two trains travelling between two stations in opposite directions, the journey requires them to share a single section of track, see Figure 1 below.

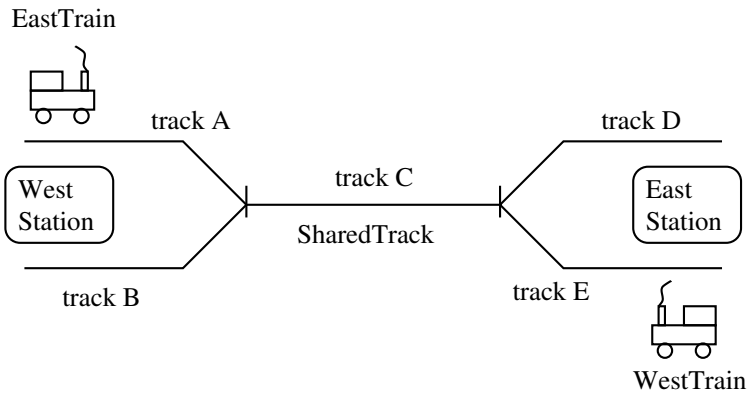

Figure 1: Railway System

- The east bound train `EastTrain` leaves the `WestStation` travels over three sections of track *A*, *C* and *D* to arrive at the `EastStation`.
- The west bound train `WestTrain` leaves the `EastStation` travels over three sections of track *E*, *C* and *B* to arrive at the `WestStation`.
- To avoid a crash the trains must be given **mutually exclusively** use of the shared section of track, i.e. *C*.
- The system consists of the **two** train processes and the **one** shared section of track process.

(a) Define three Finite State Process (FSP) language processes to model the `EastTrain`, `WestTrain` and `SharedTrack`.

Then using these three processes define a composite process that models the trains and then the complete railway system. **[24 marks]**

(b) Give the *structure* diagram for the complete Railway system. **[5 marks]**

(c) Explain how you have ensured that the two train processes `EastTrain` and `WestTrain` have *mutually exclusive* access to the shared rail track section. **[4 marks]**

Question 3

(a) In Java, concurrent programming is achieved by using *threads*. In a Java program a thread can be created using two different approaches, describe these approaches. Illustrate your answer by means of suitable Java code fragments. Explain why these two approaches are needed. **[9 marks]**

(b) Describe and explain the life-cycle states of a Java thread. Illustrate your answer by means of a diagram illustrating the relationships between the states and suitable Java code fragments that cause the state transitions. **[18 marks]**

(c) The Java API provides the **ThreadGroup** class for managing groups of threads. Give a brief explanation of the relationship between threads and thread groups. Briefly discuss how well the *thread group* feature has been implemented in Java. How would you improve it? **[6 marks]**

[TOTAL 33]

Question 4

(a) In 1974 the computer scientist C.A.R. Hoare proposed the "standard" definition of the concurrent programming concept known as a *monitor*. Describe the main features of a monitor. **[5 marks]**

(b) The Java programming language supports the concept of a monitor. Describe in detail how the monitor concept has been achieved in Java. Illustrate your answer by means of fragments of Java code. **[17 marks]**

(c) With reference to the Java program given in Appendix A.

(i) Describe the sequence of states of the object `mb` and the threads `p` and `r` during their execution; assuming that `p` calls the `post()` method **before** `r` calls the `retrieve()` method. **[6 marks]**

(ii) If the `MessageSystem` class created several `Poster` threads and several `Retriever` threads rather than just one of each, deadlock could occur.

Assuming deadlock has occurred, explain what has happened to the `Poster` and `Retriever` threads. What is the simplest change that could be made to the two `MessageBoard` methods `post()` and `retrieve()` that would stop this happening. **[5 marks]**

[TOTAL 33]

Question 5

(a) One of the first specialised concurrent programming mechanisms invented was the *semaphore*. Describe the features of semaphores. **[9 marks]**

(b) What is the *Dining Philosophers* problem (for 5 Philosophers)? Explain how *deadlock* can occur and how it can be avoided by the introduction of a *Butler*. **[6 marks]**

(c) You are given a Java class that correctly implements a `Semaphore`; where the constructor has the following form:

```
Semaphore( int max_value, int initial_value )
```

Use this semaphore class to write a Java program for the Dining Philosophers problem which avoids deadlock by using a Butler.

You must provide a `Philosopher` class, which represents the behaviour of a philosopher and a `DiningPhilosopher` class which creates the philosophers, forks and butler.

Note: you may assume that all threads execute forever. **[18 marks]**

[TOTAL 33]

Appendix A

Program for Question 4: comprises four classes `Poster`, `Retriever`, `MessageBoard` and `MessageSystem`.

```
1    class Poster extends Thread
2    {
3      private final MessageBoard messageboard;
4
5      public Poster(MessageBoard mb)
6      {
7          messageboard = mb;
8      }
9
10     public void run()
11     {
12        messageboard.post( new String("Hello mate.") );
13        messageboard.post( new String("Good Luck.") );
14     }
15
16   }
17
18
19   class Retriever extends Thread
20   {
21      private final MessageBoard messageboard;
22
23      public Retriever(MessageBoard mb)
24      {
25          messageboard = mb;
26      }
27
28      public void run()
29      {
30         Object message = null ;
31
32         message = messageboard.retrieve();
33         message = messageboard.retrieve();
34      }
35   }
```

[Continued Overleaf]

```
36  class MessageBoard
37  {
38    private Object  message = null;
39    private boolean message_posted = false;
40
41    public synchronized Object retrieve()
42    {
43       while ( !message_posted ) {
44          try {
45               wait();
46          } catch(InterruptedException e){ }
47       }
48       message_posted = false;
49       notify();
50       return message ;
51    }
52
53    public synchronized void post(Object new_message)
54    {
55       while ( message_posted ) {
56          try {
57               wait();
58          } catch(InterruptedException e){ }
59       }
60       message = new_message;
61       message_posted = true;
62       notify();
63    }
64  }
65
66  class MessageSystem
67  {
68    public static void main(String args[]) {
69        MessageBoard mb = new MessageBoard() ;
70        Poster        p = new Poster(mb) ;
71        Retriever     r = new Retriever(mb) ;
72
73        p.start();
74        r.start();
75    }
76  }
```