UNIVERSITY OF WESTMINSTER

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Module Title: **Concurrent Programming**

Module Code: **6SENG002W, 6SENG004C**

Exam Period: **January 2020**

Time Allowed: **2 Hours**

INSTRUCTIONS FOR CANDIDATES

PLEASE WRITE YOUR STUDENT ID CLEARLY AT THE TOP OF EACH PAGE.

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

Answer THREE questions.

Each question is worth 33 marks.

Only the THREE questions with the HIGHEST MARKS will count towards the FINAL MARK for the EXAM.

THIS PAPER MUST NOT BE TAKEN OUT OF THE EXAMINATION ROOM

DO NOT TURN OVER THIS PAGE UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO

Question 1

Given the following Finite State Process (FSP) processes and parallel systems for a simple version of the Producer/Consumer problem:

```
const N    = 3
range DATA = 1..N
set   IN   = { in[DATA] }

Producer_1 = ( produce[1] -> in[1] -> Producer_1 ) .

Producer_2 = ( produce[2] -> in[2] -> Producer_2 ) +IN .

BUFFER = ( in[ i : DATA ] -> out[ i ] -> BUFFER ) .

Consumer = ( out[ x : DATA ] -> consume[x] -> Consumer ) .


|| SYSTEM_1 = ( Producer_1 || BUFFER || Consumer ) .

|| SYSTEM_2 = ( Producer_2 || BUFFER || Consumer ) .
```

Given the above FSP process definitions:

(a) What is the FSP operator "`+IN`" used in definition of `Producer_2`? What effect does it have on `Producer_2`? **[5 marks]**

(b) Based on the FSP definition of the action "`out[ x : DATA ]`" used in the definition of `Consumer`, give the fully expanded version of the `Consumer` process.

Draw the state machine diagram (LTS graph) for the `Consumer` process. **[7 marks]**

(c) Give the alphabets of the following processes:

- `Producer_1`
- `Producer_2`
- `BUFFER`
- `Consumer`

[7 marks]

[Continued Overleaf]

(d) **(i)** Draw the *Alphabet* diagram for `SYSTEM_1`. **[5 marks]**

(ii) Draw the *Alphabet* diagram for `SYSTEM_2`. **[5 marks]**

(iii) Describe the difference between how the `BUFFER`'s actions are performed, either asynchronously, synchronously or blocked, in the two systems. **[4 marks]**

[TOTAL 33]

Question 2

A bank's ATM ("Cash Machine") is used by two customers Jack and Jill; the ATM is refilled with cash by Nigel the bank's cashier. The specification of the system is as follows:

- The ATM has a maximum limit of cash.
- The ATM accepts withdrawal requests for cash from its customers.
- To ensure that an accurate record of the cash in the ATM is maintained, the two customers and the cashier must have **mutually exclusive** access to the shared ATM.
- Once a customer has mutually exclusive control of the ATM he or she can withdrawal an amount of cash from the ATM. If the ATM has enough cash then it will dispense the amount of cash requested and reduce the cash in the ATM, otherwise it indicates that it does not have enough and no cash is dispensed, leaving the cash in the ATM unchanged.
- Once the cashier (Nigel) has mutually exclusive control of the ATM he can refill the ATM with cash to its limit.
- The two customers Jack and Jill perform one withdrawal each.

(a) Using the Finite State Process (FSP) language define processes to model the following:

- `ATM`
- `Customer`
- `Cashier`

[23 marks]

(b) Using your three processes, define a composite process that models the ATM system comprising 4 processes: 2 customers, 1 cashier and 1 ATM. **[4 marks]**

(c) Describe how you have used FSP features to ensure that the two customer processes (Jack and Jill) and the one cashier process (Nigel) have *mutually exclusive* access to the ATM. **[6 marks]**

[TOTAL 33]

Question 3

(a) In Java, concurrent programming is achieved by using multiple *threads*. In a Java milt-threaded 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 the Java language designers included two different approaches to create a thread rather than just one. **[9 marks]**

(b) Describe and explain the life-cycle of a Java thread. Illustrate your answer by means of a diagram illustrating the various thread states and the relationships between them. Include in your description examples of Java code fragments that cause a thread to move from one state to another. **[18 marks]**

(c) The Java APO provides the `ThreadGroup` class. Explain its purpose and give a brief explanation of the relationship between threads and thread groups.

Further, discuss how well the `ThreadGroup` class achieves its aim. Can you suggest any additional features that could be added to it? **[6 marks]**

[TOTAL 33]

Question 4

Appendix A contains a Java program that models posting and retrieving messages to and from a message board.

(a) In the early 1970s Per Brinch-Hansen invented the concurrent program structuring mechanism known as a *monitor*. Following on from this in 1974 Tony Hoare proposed an improved version of a monitor that became the "standard" definition of a *monitor*. Describe the main features of a monitor. **[5 marks]**

(b) The Java language supports a version of the *monitor* mechanism. Describe in detail how *monitors* have been implemented in Java. Your answer should be illustrated by reference to the program code given in Appendix A. **[16 marks]**

(c) With reference to the 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. **[7 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) Describe the features of the concurrent programming mechanism introduced by E. W. Dijkstra in the mid 1960s, known as a *semaphore*. **[9 marks]**

(b) Using Java's `java.util.concurrent.Semaphore` class, give suitable Java code fragments illustrating how semaphores can be used to achieve *ordering the actions* of two Java threads. Assume that the two threads perform the following print actions in their `run()` methods:

```
Thread T1:  System.out.println( "T1: Action 1") ;
            System.out.println( "T1: Action 3") ;

Thread T2:  System.out.println( "T2: Action 2") ;
            System.out.println( "T2: Action 4") ;
```

Your Java code must illustrate how to achieve the following output sequence:

```
T1: Action 1
T2: Action 2
T1: Action 3
T2: Action 4
```

[18 marks]

(c) With reference to your Java code given in answer to part **(b)**; explain how you have used semaphores to achieve the action ordering. **[6 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         super() ;
8         messageboard = mb;
9      }
10
11     public void run()
12     {
13        messageboard.post( new String("Hello mate.") );
14        messageboard.post( new String("Good Luck.") );
15     }
16   }
17
18   class Retriever extends Thread
19   {
20      private final MessageBoard messageboard;
21
22      public Retriever(MessageBoard mb)
23      {
24         super() ;
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  }
```

END OF THE EXAM PAPER