Concurrent and Distributed Systems (2018-19)

Assignment 1 (50% of Total CW Marks)

Submission Deadline: Monday Week 7 (11 March 2018 at noon)
Feedback: 15 working days after the submission

Submission Instructions:

This coursework has two elements related to it, the first one is a design report (including Pseudo-code semaphore solution) and the second one is an implementation for which output screenshots and code listings are required in the report.

The report, in PDF format, should be submitted to turn-it-in via Blackboard on or before the deadline.

The working implementation should be demonstrated during the laboratory session on week 8.

Question 1 – Design of Concurrent Systems [50 Marks]

A toy train circuit (figure of eight) is shown in Figure 1. Trains of type A enter via section 0 and rotate clockwise around block A. Trains of type B enter via section 16 and rotate clockwise around block B. Trains of type C enter via section 17 and go through 18, 9, 8, 7, 19, and 20. (Note. All the trains have to share sections 7, 8 and 9).

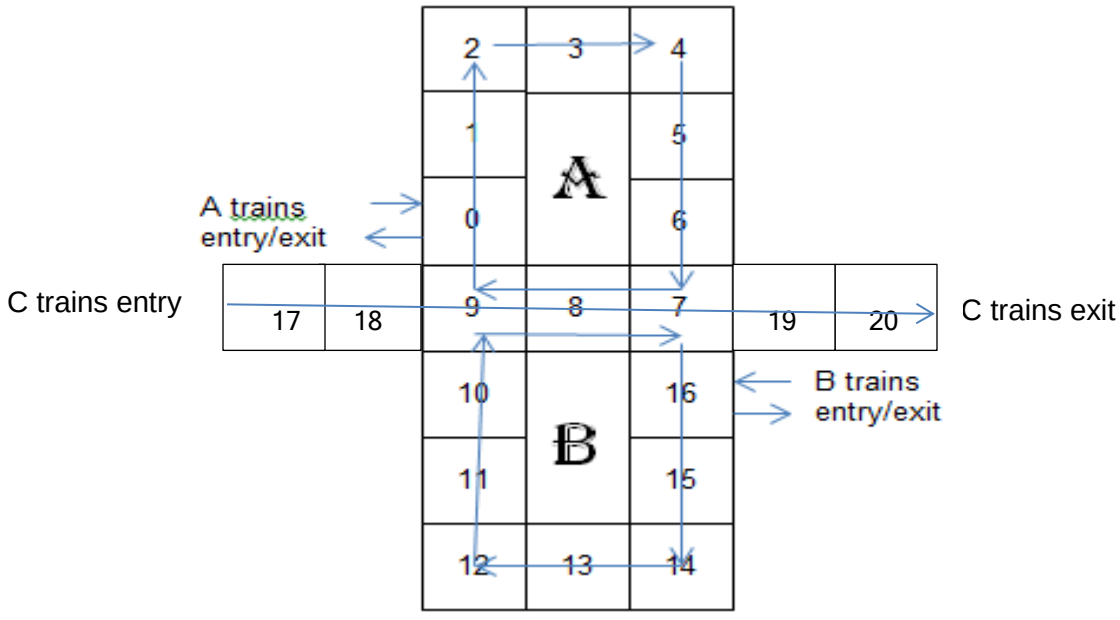

Figure 1: Toy Train Circuit

A train joining the circuit must first make sure the section ahead of them is free (e.g. A train entering via section x must ensure section x is not occupied by another train at that time). To avoid train crashes only one train at any time can be in any of the available sections. (each train fits into one section of the track)

Assuming that shared memory is available:

(a) Design a solution using **semaphores**.

(b) Document any design decision made, issues and/or problems encountered and rationale for the approach taken to solve the problem.

Assessment Criteria for Question 1:

(i) correct and logical use of semaphores statements. ***(5 marks)***
(ii) Clear and efficient design and appropriate use of pseudo-code, programming structures, instructions, etc. ***(25 marks)***
(iii) Quality of documentation of problems encountered. ***(5 Marks)***
(iv) Justification of approach taken to solve the problem. ***(15 marks)***

Question 2 - Programming Concurrent Systems [50 Marks]

Implement the design developed in Question 1 into a Java based solution using **semaphores** and test it. Ideally the simulation should show concurrent activity with five trains of each type joining/leaving the circuit. Each train of either type A or B should do five loops before exiting.

The resulting output should show snapshots of the trains currently on the circuit as the train movement occurs. This output should be sufficiently informative to allow the train traffic behaviour to be determined.

(**Note**. The *java.util.concurrent* library has the *Semaphore* class which you may use; or you may use the class on the next two pages which also represents a *Semaphore*, which avoids you having to include any *try/catch* statements in your code. It also uses the traditional *P()* and *V()* operations instead of *acquire()* and *release()* operations respectively.)

[50 marks]

Assessment Criteria for Question 2 :

(i) logically correct implementation of semaphore solution in Java ***(20 marks)***
(ii) clear efficient well documented code ***(5 marks)***
(iii) quality of program testing ***(10 marks)***
(iv) Informative comment on problems/issues encountered. ***(5 marks)***
(v) Demo of the system ***(10 marks)***

```
import java.util.concurrent.*;
//MageeSemaphore.java
//This is an implementation of the traditional (counting) Semaphore with P() and V() operations
class MageeSemaphore
{
   private Semaphore sem;

   public MageeSemaphore (int initialCount)
   {
            sem = new Semaphore(initialCount);
   } // end constructor

   public void P()
   {
            try {
               sem.acquire();
            } catch (InterruptedException ex) {System.out.println("Interrupted when waiting");}
   } // end P()

   public void V()
   {
            sem.release();
   } // end V()

} // end MageeSemaphore
```

To help record and display the train track activity you may wish to use the following class:

```
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.Iterator;
// Represents the train track activity in a thread-safe CopyOnWriteArrayList<String>
// called theActivities
// - addMovementTo(<Integer>) adds a train movement (destination) activity to the record
// - addMessage(<String>) adds a message to the record
// - printActivities() display all the activity history of the train movement
// - trackString() takes a snapshot of the traintrack (with trains) for printing
public class Activity {

    private final CopyOnWriteArrayList<String> theActivities;

    private final String[] trainTrack;

    // Constructor for objects of class Activity
    // A reference to the track is passed as a parameter
    public Activity(String[] trainTrack) {
        theActivities = new CopyOnWriteArrayList<>();
        this.trainTrack = trainTrack;
    }

    public void addMovedTo(int section) {
        // add an activity message to the activity history
        String tempString1 = "Train " + trainTrack[section] + " moving/moved to [" + section + "]";
        theActivities.add(tempString1);
        // add the current state of the track to the activity history
        theActivities.add(trackString());
    }// end addMovedTo

    public void addMessage(String message) {
        // add an activity message to the activity history
        String tempString1 = message;
        theActivities.add(tempString1);
```

```
    }// end addMessage

    public void printActivities() {
        // print all the train activity history
        System.out.println("TRAIN TRACK ACTIVITY(Tracks [0..20])");
        Iterator<String> iterator = theActivities.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }// end printActivities

    // Utility method to represent the track as a string for printing/display
    public String trackString() {
        String trackStateAsString = trainTrack[2] + trainTrack[3] + trainTrack[4] + "\n"
                + trainTrack[1] + "    " + trainTrack[5] + "\n"
                + trainTrack[0] + "    " + trainTrack[6] + "\n"
                + trainTrack[17] + "    " + trainTrack[18] + "\n"
                + trainTrack[9] + trainTrack[8] + trainTrack[7] + "\n"
                + trainTrack[19] + "    " + trainTrack[20] + "\n"
                + trainTrack[10] + "    " + trainTrack[16] + "\n"
                + trainTrack[11] + "    " + trainTrack[15] + "\n"
                + trainTrack[12] + trainTrack[13] + trainTrack[14] + "\n";
        return (trackStateAsString);
    }// end trackString

}// end Activity
```