THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Information and Physical Sciences
Semester 2, 2022 - COMP2240/COMP6240 - Operating Systems

Assignment 2 (15%)

Submit using Canvas by **11:59 pm, Friday 16th September 2022**

version 1.0.0

Note: Problems 1, 2, and 3 are for both COMP2240 & COMP6240 students. Problem 4 is only for COMP6240 students.

1. Tasks:

1.1. Problem 1: Sharing the Bridge

A new single lane bridge is constructed to connect the North Island to the South Island of New Zealand. Farmers from each island use the bridge to deliver produce to the other island, return back to their island and this is repeated indefinitely. It takes a farmer carrying produce 20 steps to cross the bridge. Once a farmer (say a North Island farmer identified as `N_Farmer1`) crosses the bridge (from North Island to South Island) he just attempts to cross the bridge in the opposite direction (from South Island to North Island) and so on. Note that the ID of the farmer does NOT change.

The bridge can become deadlocked if a northbound and a southbound farmer are on the bridge at the same time (New Zealand farmers are stubborn and will not back up). The bridge has a large neon sign above it indicating the number of farmers that have crossed it in either direction. The neon sign counts multiple crossing by the same farmer.

Using **semaphores**, design and implement an algorithm that prevents deadlock. Use **threads** to simulate multiple/**concurrent** farmers and assume that the stream of farmers are constantly attempting to use the bridge from either direction. Your program should input parameters at runtime to initialise the number of farmers from each direction. For example `[N=5, S=5]` would indicate a constant stream of 5 farmers from each direction wanting to use the bridge. You also make sure that the solution is starvation-free (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa should not occur).

Add **20 ms delay** between every 5 steps of the farmers take.

You should **terminate the problem after NEON = 100**.

1.1.1. Sample Input/output for Problem 1

The input file will be formatted as follows:

```
N=2, S=2
```

The input indicates the program is initialized with 2 farmers from each direction who will constantly attempt to use the bridge to go from one island to other.

The (***partial***) output from one execution is as follows:

```
N_Farmer1: Wating for bridge. Going towards South
S_Farmer2: Wating for bridge. Going towards North
S_Farmer1: Wating for bridge. Going towards North
N_Farmer2: Wating for bridge. Going towards South
N_Farmer1: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 10.
N_Farmer1: Crossing bridge Step 15.
N_Farmer1: Across the bridge.
NEON = 1
S_Farmer2: Crossing bridge Step 5.
N_Farmer1: Wating for bridge. Going towards North
S_Farmer2: Crossing bridge Step 10.
S_Farmer2: Crossing bridge Step 15.
S_Farmer2: Across the bridge.
NEON = 2
N_Farmer2: Crossing bridge Step 5.
N_Farmer2: Crossing bridge Step 10.
N_Farmer2: Crossing bridge Step 15.
S_Farmer2: Wating for bridge. Going towards South
N_Farmer2: Across the bridge.
NEON = 3
S_Farmer1: Crossing bridge Step 5.
N_Farmer2: Wating for bridge. Going towards North
S_Farmer1: Crossing bridge Step 10.
S_Farmer1: Crossing bridge Step 15.
S_Farmer1: Across the bridge.
NEON = 4
N_Farmer1: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 10.
S_Farmer1: Wating for bridge. Going towards South
N_Farmer1: Crossing bridge Step 15.
N_Farmer1: Across the bridge.
NEON = 5
…
…
…
```

NOTE: For the same input the output may look somewhat different from run to run.

1.2. Problem 2: Ice-cream parlour

A new ice-cream parlour has been opened at Shortland Student Hub. The parlour does not offer take away service but there is only **five seats** in the parlour to eat in. A customer may arrive to the parlour at **any time** and may take a **certain time** to finish his ice-cream. The manager servers the customers in the order they arrives but adds a peculiar rule to it. If a customer arrives when there is an empty seat, then the customer can immediately take a seat. However, if all the five seats are occupied (i.e. there are five customers enjoying their ice-creams at any instant) then all the arriving customers have to wait for the entire party (all current customers) to leave before they can get their seats.

Using **semaphores** design and implement an algorithm that manages the customers entering and leaving the ice-cream parlour in line with manager's rules. Use **threads** to simulate multiple/**concurrent** customers. Your solution must be fair - starvation free. Assume no time is wasted in taking seat, serving/starting eating ice-cream and leaving the parlour.

1.2.1. Sample Input/output for Problem 2

The input file will be formatted as follows:

```
0 C1 5
0 C2 7
0 C3 8
2 C4 5
3 C5 5
4 C6 3
7 C7 5
10 C8 5
END
```

Where each line, except the last, contains information regarding a customer
Arrival-time Customer-ID Ice-cream-eating-time
Input ends with a line containing the word END.

You can assume that in the input the customers will be sorted in order of their arrival times.

The output should be as follows:

```
Customer    Arrives      Seats    Leaves
C1               0          0         5
C2               0          0         7
C3               0          0         8
C4               2          2         7
C5               3          3         8
C6               4          8        11
C7               7          8        13
C8              10         10        15
```

Output shows information of each customer in a separate line. Each line contains Customer-ID, Arrival-time, time when the customer seats in the ice-cream parlour and time when the customer leaves the parlour.

1.3. Problem 3: Monitoring the Ice-cream parlour

You will need to implement a solution for Problem 2 (Ice-cream parlour) using monitor.

Using **monitor**, design and implement an algorithm that manages the customers entering and leaving the ice-cream parlour in line with manager's rules. Use **threads** to simulate multiple/**concurrent** customers. Your solution must be fair and starvation free, i.e., the customers will be served in the order of their arrival time and no customer should be waiting for indefinite time. Assume no time is wasted in taking seat, serving/starting eating meals and leaving the parlour.

1.3.1. Sample Input/output for Problem 3

The sample input/output will be same as shown in Problem 2.

1.4. Problem 4: Additional Task for COMP6240 students only [NOT for COMP2240 students]

In the lecture, two algorithms, namely Dekker's algorithm and Peterson's algorithm, were discussed as software approaches to mutual exclusion. A couple of other algorithms exist in the literature for the same purpose. You are asked to perform a survey on the software approaches to mutual exclusion and prepare a report on that. Your survey should include at least four different algorithms (*may and may not include the above two*). You should discuss the design principle, suitability of generalisation for n processes, suitability for multiprocessor/multi-core environment, relative advantages/disadvantages.

Your survey report should be **between 4 and 6 pages** (*single space*) of content, excluding cover and references.

2. Other Submission Requirements:

Your submission must also conform to the follow considerations.

2.1. Programming Language:

The programming language is Java, versioned as per the University Lab Environment (*Note this appears to have been **rolled back on the Lab machines**, which means we're **currently targeting a subversion of Java 11 – so please be conscious of this***). You may only use standard Java libraries as part of your submission.

2.2. User Interface:

The output should be printed to the console, and strictly following the output samples given in the assignment package. While there are no marks allocated specifically for the Output Format, there will be a deduction when the result format varies from those provided.

2.3. Input and Output:

Your program will accept data from an input file of name specified as a command line argument. The sample files `P1-1.txt` and `P2-1.txt` (*containing inputs for Problem 1 and 2/3 respectively*) are provided to demonstrate the required input file format. **Hint:** the **Java Scanner Library** is something you will likely want to use here!

Your submission will be tested with the above data and will also be tested with other input files.

Your program should output to standard output (*this means output to the Console*). Output should be strictly in the given format (*see the output files below*).

The sample files `P1-1out.txt` and `P2-1out.txt` (*containing output for* `P1-1.txt` *and* `P2-1.txt` *respectively*) are provided to demonstrate the required output (and input) format which **must be strictly maintained**. **If output is not generated in the required format then your program will be considered incorrect.**

2.4. Mark Distribution:

Mark distribution can be found in the assignment feedback document (`Assign2Feedback2240.pdf` and `Assign2Feedback6240.pdf`).

2.5. Deliverable:

1. Your submission will contain your program source code with documentation and the report (*below*) in the root of the submission. These files will be zipped and submitted in an archive named **`c9876543.zip`** (*where* **`c9876543`** *is your student number*) – do not submit a `.rar`, or a `.7z`, or etc.
2. Your **main classes** (for each different problem) will be named **`P1.java`**, **`P2.java`** and **`P3.java`** respectively, and your program will thus compile with the command **`javac P1.java`**, **`javac P2.java`** and **`javac P3.java`** respectively.

   Your program will be executed by running **`java P1 input.txt`**, **`java P2 input.txt`** and **`java P3 input.txt`** respectively.

   …where **`input.txt`** can be *any* filename – **do not hardcode filenames or extensions**!

   Note: If your program can not be compiled and executed in the expected manner (*above*) then please add a `readme.txt` (containing any special instructions required to compile and run the source code) in the root of the submission. Please note that such non-standard submissions will be **marked with heavy penalty**.
3. Brief **1 page** (*A4*) report of the how you tested your programs to ensure they enforced mutual exclusion and are deadlock and starvation free. Specifically, your report should include discussion on edge cases you considered and behaviour of your algorithm on those cases, any specific trick/technique you applied and did you face any specific issue.
4. COMP6240 students will in addition to the above requirements also submit their survey report additional task as a **PDF document**, called SurveyReport.pdf, and also placed in the root of the submission archive.

2.6. Submission Notes:

a) Assignments submitted after the deadline (**11:59 pm 16th September 2022)** will have the final result reduced by 10% of the maximum possible marks per day. A 'day' begins at 12:00 Midnight, and once this time is passed, the penalty for the whole day is applied. This is UoN Policy.

b) If your assignment is not prepared and submitted following above instructions then you will lose most of your marks despite your algorithms being correct.

Nasim and Dan
v1.0 2022-08-25