Operating Systems Assignment: Semaphores

1. Introduction

This assignment has the objective of developing your understanding of the use of semaphores for controlling access to resources such as often occurs in an operating system.

2. The scenario

A company has its headquarters in the city centre of Cape Town. It employs M people who need to travel between the headquarters and its N-1 other branches during the day to do some work there. The headquarters is also a branch, and some people do some work there as well. So there are N branches in total.

The company has one taxi to help with the transport. Every morning the taxi starts to transport passengers from the headquarters (branch 0) to the other branches. The taxi is big enough to carry all M people. At 9am they all arrive at the headquarters to clock in. During the day they do some work at the branches, using the taxi to move between them.

When a person wants to use the taxi to go to another branch, they press a hail button to hail the taxi, wait for the taxi to arrive and open its doors, enter the taxi, tell the driver to which branch they need to go, and wait for the taxi to arrive at that branch.

In deciding where to pick up passengers next, the taxi driver tries to be fair. The taxi goes outbound until it reaches the furthest branch to pick up or drop off a passenger. Then it goes inbound until it reaches the branch closest to the headquarters (which may be the headquarters) to pick up or drop off passengers. While driving outbound and inbound like this, the taxi picks up and drops off passengers along the way. That is, the taxi does not reverse direction until it has gone as far in one direction as needed to pick up and/or drop off passengers.

If a person hails the taxi and it arrives, they get in, regardless of which direction the taxi is going (that is, whether it's going inbound or outbound).

Having arrived at a branch, the taxi remains there if (i) there are no passengers to be picked up, (ii) there are no passengers waiting to be dropped off at another branch, and (iii) it has not been hailed by passengers elsewhere.

The taxi takes two minutes to move from one branch to the next (it's a very fast taxi). If the driver needs to stop at a particular branch to pick up or discharge passengers, it takes one minute to do so. After a person disembarks, that person cannot board the taxi again until the taxi leaves and returns to that branch, unless the taxi is idle (no Hail button was pushed and no one has told the driver where to take them next) right after that person disembarks.

2. The task

Your assignment is to write a Java program, called *Simulator.java*, that uses Java threads, to simulate the movement of people between branches using the taxi. You will use semaphores to synchronize the taxi thread and the people threads.

Your program needs to have a *Taxi* class from which a Taxi object containing the taxi thread is instantiated, and a *Person* class from which Person objects containing a person thread each are instantiated. In either the *Taxi* class or some other coordinator class(es), place all the semaphores and other state variables as private data fields.

The main method in the *Simulator* driver class starts everything going.

A Person object calls a method in the Taxi object to indicate the person is waiting at a branch to be picked up. The Person thread blocks on a semaphore inside the method until picked up by the taxi. When picked up, the Person object calls another Taxi method to indicate the desired branch number. The Person thread blocks on a semaphore inside the method until the taxi arrives at the branch. When arriving at a branch, the Taxi thread waits for one minute to let passengers get on and off, then the Taxi thread blocks until all passengers have safely (dis)embarked.

Each Person object will operate according to an input script which determines which branch the person goes to and how long the person works at that branch before using the taxi to go to another branch.

2.1 Input

The input data to your Java program consists of a file containing the number of people using this service, the number of branches on the route, and the work/travel patterns for each person. The filename will be provided on the command line:

```
% java Simulator <filename>
```

The file format is as follows:

<number of people> <newline>
<number of branches> <newline>
{<person number> (<branch, duration>) { , (<branch, duration>)} <newline> }

An expression of the form '{<n>}' indicates that the element <n> is repeated one or more times.

A duration is expressed as a quantity of minutes.

Example:

```
3
5
0 (1, 10), (0, 5), (3, 40)
1 (2, 15), (1, 23), (2, 18), (4, 5), (3, 50)
2 (3, 100)
```

2.2 Output

Your program should produce (print) a trace of the simulation that indicates when the taxi arrives at a branch or leaves a branch, and when a person hails the taxi, embarks (and requests a destination) or disembarks.

A trace statement will take the following form:

<trace statement> ::= <time> branch <branch number>: <entity id> <event description>

The statement identifies the time, the number of the branch at which the event occurred, the entity reporting the event, and the event itself.

An entity ID identifies either the taxi, or a person. An event description indicates the type of event:

<time> ::= <digit>:<digit><digit>
<entity id> ::= taxi | person <person number>
<event description> ::= hail |request <destination branch> | disembark | arrive | depart

A time value is given in 24-hour clock format.

Example:

```
…
10:30 branch 0: person 1 hail
10:35 branch 3: taxi depart
10:37 branch 0: taxi arrive
10:37 branch 0: person 1 request 3
10:37 branch 2: person 2 hail
10:38 branch 0: taxi depart
…
```

2.3 Time

As your program implements a simulation, it should not function in real-time i.e. a simulated second should not correspond to an actual second.

In the case of assignment one, time was entirely simulated. Actions were given a weighting in terms of time units and the 'clock' (a counter) was advanced to reflect actions that had taken place. For this assignment. You COULD attempt to do the same, however, a simpler approach is to apply a scale:

- One minute of simulated time is equal to 33 milliseconds of real time.

Thus, when you want the taxi to take two minutes to 'travel' from one branch to another, you would use the Thread class *sleep()* method as follows:

```
…
Thread.sleep(17*2);
…
```

This scaling produces a working (eight hour) day roughly equivalent to 15.84 seconds.

2.4 Testing

For some stress testing, you should consider testing your program with the following kinds of input data:

- One person who works for a long time at each stop so the taxi goes into its idle state.
- Many people working for short periods of time, keeping the taxi busy.

3. Marking and submission

Your work will be manually marked, however, you should submit your materials for assessment via the automatic marker within a single ‘.zip’ file bearing your student number.

Primarily, assessment concerns the successful implementation of person and taxi synchronisation:

- Does a person hail the taxi, and the taxi respond?
- Does the taxi move to the correct branch when hailed?
- Is (dis)embarkation behaviour correct?
- Does a person get to the branch requested?
- Etc.

However, assessment will be conducted by running your simulator on a variety of test inputs, so it needs to adequately handle I/O aspects to enable this to happen.

END