Concurrent Programming

Exercise Booklet 5: Semaphores (cont)

Solutions to selected exercises (◊) are provided at the end of this document. Important: You should first try solving them before looking at the solutions. You will otherwise learn **nothing**.

Exercise 1. (◊) On Fridays the bar is usually full of Jets fans. Since the owners are Patriots fans they would like to implement an access control mechanism in which one Jets fan can enter for every two Patriots fans.

1. Implement such a mechanism, assuming that Jets fans will have to wait indefinitely if no Patriots fans arrive. You may assume, to simplify matters, that once fans go in, they never leave the bar.
2. Modify the solution assuming that, after a certain hour, everybody is allowed to enter (those that are waiting outside and those that yet to arrive). For that there is a thread that will invoke, when the time comes, the operation `itGotLate`. You may assume that the code for this thread is already given for you, the only thing that you must do is define the behavior of `itGotLate` and modify the threads that model the Jets and Patriots fans.

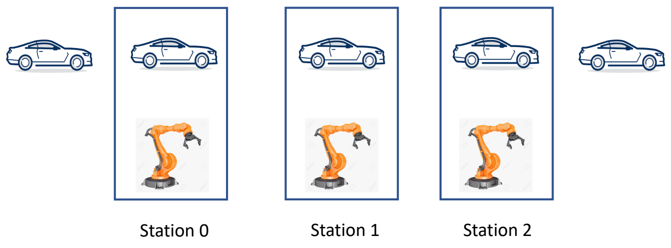
Exercise 2. In the office there is a unisex restroom with n toilets. The restroom can be used by both men and women, but cannot be used by men and women at the same time. Provide a solution using semaphores. The solution should be free from deadlock but not necessarily from starvation.

Exercise 3. Model a ferry between two coasts, say the East (0) and the West (1) coasts, using semaphores. The ferry has capacity for N passengers and works in the following way. It waits at one coast until it fills up to capacity and then automatically switches to the other coast. When it arrives at a coast, it waits for all the passengers to get off and then allows new passengers to board. The ferry and each passenger has to be implemented as a thread. There is no cap on the number of passengers at either coast. Also, for the purpose of simplicity, passengers use the service once and then never again.

Exercise 4. In a gym there are four apparatus, each involving a different muscle group. The apparatus are loaded with weight discs (all of the same size). Each gym client has a routine that indicates what apparatus she must use, in what order and with what weight (the routine could include repeating the use of an apparatus). The gym requires that each client, when finished using an apparatus, unloads all weight discs and places them in their storage area (this includes consecutive uses of the same apparatus).

1. Write code that simulates the gym's workings, guaranteeing mutual exclusion in the access of the shared resources and freedom of deadlock.
2. Indicate whether your solution is free of starvation. If it's not, indicate how you could obtain it.

Exercise 5. We would like to model a *control system* for an automatic car wash. Each car traverses three phases: blast, rinse and dry. Each of these phases is executed by a machine. All vehicles follow these three phases in that exact order.



Some additional considerations:

- A machine can only start working on a car once the car is in place
- A car can only leave a phase once it knows the machine has finished its work
- There can be at most one car in each phase
- A car cannot advance to the next phase if it is occupied by another car

Model the cars and each machine with appropriate threads.

Exercise 6. Blood donations are received in a hemotherapy center. Extraction takes place in one of 4 beds and takes a random amount of time. People that arrive to donate are received in the order in which they arrive, but if there are no free beds they wait in the waiting room. In the waiting room there are 10 outdated magazines that people read while they wait for their turn. While waiting donors continuously pick up a magazine, read it, put it down. When a bed is freed, the next donor is called from the waiting room, who leaves the magazine she was reading. As soon as a magazine has been freed, the donors that are waiting compete to take it.

Provide a solution using semaphores that models this scenario. Hint: have the process of acquiring a magazine and that of acquiring a bed be the concurrent threads that are spawned for each donor.

Exercise 7. Consider the following simplified version of "The Settlers" game. It consists in managing a village by means of villagers and numerous resources.

The player has 8 villagers to whom she assigns a list of tasks that are executed indefinitely. The tasks that may be executed are:

- extract a resource from a site
- deposit a resource in a site
- Move from one site to another – this takes a villager 10 seconds

In this version of the game the village has the following sites:

- Lake, where water can be extracted
- Farm, where 10 seconds after watering a parcel of land, food can be extracted
- Warehouse, where water and food can be stored for future extraction

Bear in mind that the extraction task requires that the villager wait until the resources are available. If there are none available, the villager must wait: in the case of the farm for 10 seconds after she has watered the parcel, in the case of the warehouse until there is a resource to extract.

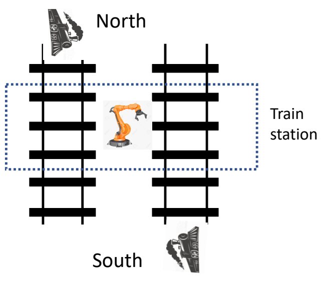
Model this scenario using semaphores, assuming that the warehouse has infinite space and there are an infinite number of parcels in the farm.

Exercise 8. (◊) Model a vehicle crossing between two endpoints. We'll denote these endpoints 0 and 1. Since the crossing is narrow, it does not allow for vehicles to travel in opposite directions. Your solution must allow multiple vehicles to use the crossing so long as they are travelling in the same direction. Use a thread `Vehicle(myEndpoint)` to model a vehicle located at endpoint `myEndPoint` (0 or 1).

- How would you modify your solution so that at most 3 vehicles are on the crossing at any given time?
- Is your solution fair?

Exercise 9. Trains run in both North-South (0) and South-North (1) direction, each on its own track.

Two kinds of trains ride these tracks: passenger trains and freight trains. You are to model the behavior of a train station for each of these two kinds of trains.



- Passenger trains: A passenger train can only stop at the station if there are no other trains on the same track. It does not matter whether there is a train at the station on the track corresponding to trains travelling in the opposite direction.
- Freight trains: The station has the ability to load freight trains via a loading machine. In order for a freight train to be able to be loaded, there must not be trains at the station (in any of the two tracks). Moreover, if the freight train makes use of the station, it cannot leave until the loading machine is done.

Model the passenger train and the freight train as threads. The loading machine has already been modeled for you below. Additional semaphores may be required.

```
// declarations
Semaphore permToLoad = new Semaphore(0);
Semaphore doneLoading = new Semaphore(0);
// complete

Thread.start { // PassengerTrain
    // complete
}

Thread.start { // FreightTrain
    // complete
}

Thread.start { // LoadingMachine
   while (true) {
      permToLoad.acquire();
      // load freight train
      doneLoading.release();
  }
}
```

1 Solutions to Selected Exercises

Answer to exercise 1

Groovy

```
import java.util.concurrent.Semaphore;

Semaphore ticket = new Semaphore(0);
Semaphore mutex = new Semaphore(1);

20.times{
    Thread.start { // Patriots
     ticket.release();
    }
}

20.times {
Thread.start { // Jets
    mutex.acquire();
    ticket.acquire();
    ticket.acquire();
    mutex.release();
}
}
```

```
import java.util.concurrent.Semaphore;

Semaphore ticket = new Semaphore(0);
Semaphore mutex = new Semaphore(1);
boolean itGotLate = false;

Thread.start { // Jets
    mutex.acquire();
    if (!itGotLate) {
       ticket.acquire();
       ticket.acquire();
    }
    mutex.release();
}

Thread.start { // Patriots
    ticket.release();
}

def itGotLate() {
    itGotLate=true;
    ticket.release();
    ticket.release();
}

return
```

Java

```
package basics;
import java.util.concurrent.Semaphore;
import javax.swing.plaf.multi.MultiTextUI;

public class Bar {

    static Semaphore ticket = new Semaphore(0);
    static Semaphore counters = new Semaphore(1);
    static int jets=0;
    static int patriots=0;

    public static class Jet implements Runnable {

        static Semaphore mutex = new Semaphore(1);

        public void run() {

            try {
                mutex.acquire();
                ticket.acquire();
                ticket.acquire();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            try {
                counters.acquire();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            jets++;
            assert jets*2<=patriots;
            System.out.println("J");
            counters.release();
            mutex.release();
        }
    }

    public static class Patriot implements Runnable {

        public void run() {
            try {
                counters.acquire();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            ticket.release();
            patriots++;
            System.out.println("P");
            counters.release();
        }
    }

    public static void main(String[] args) {
        for (int i=0; i<20; i++) {
            new Thread(new Jet()).start();
        }

        for (int i=0; i<20; i++) {
            new Thread(new Patriot()).start();
        }
    }
}
```

Answer to exercise 8

```
import java.util.concurrent.Semaphore;

Semaphore useCrossing = new Semaphore(1); //mutex
endpointMutexList = [new Semaphore(1, true), new Semaphore(1, true)]; // Strong sem.
noOfCarsCrossing = [0,0]; // list of ints
r = new Random();

100.times { // spawn 100 cars
    int myEndpoint = r.nextInt(2);  // pick a random direction
    Thread.start {
      endpointMutexList[myEndpoint].acquire();
      if (noOfCarsCrossing[myEndpoint] == 0)
        useCrossing.acquire();
      noOfCarsCrossing[myEndpoint]++;
      endpointMutexList[myEndpoint].release();

      //Cross crossing
      println ("car $it crossing in direction "+myEndpoint + " current totals "+noOfCarsCrossing);

      endpointMutexList[myEndpoint].acquire();
      noOfCarsCrossing[myEndpoint]--;
      if (noOfCarsCrossing[myEndpoint] == 0)
        useCrossing.release();
      endpointMutexList[myEndpoint].release();
```

```
        }
}
```