Sistemas Distribuídos

Algorithmic Analysis of the Airport Problem

António Rui Borges

Summary

- *Interaction among entities*
  - *Porter : passenger interaction*
  - *Passenger : passenger interaction*
  - *Bus driver : passenger interaction*
- *Role of the General Repository of Information*
- *Code organization*
- *Code validation*

Interactions among the entities - 1

There are three types of interaction among the entities

- *porter* : *passenger*
  - the last passenger to reach the arrival lounge gives notice to the porter he must start unloading the plane’s hold
  - when the porter places on the conveyor belt a bag from a passenger who has the airport as final destination, he must alert her that a bag she owns is there
  - when the porter has finished unloading the bags from the plane’s hold, he must alert all possible waiting passengers that there are no more bags to be collected
- *passenger* : *passenger*
  - the last passenger to reach either the arrival, or the departure, terminal, must give notice that all of them may go home, or proceed to check in for the next leg of the journey

Interactions among the entities - 2

- *bus driver : passenger*
  - a passenger who is going to queue to board the bus, should alert the bus driver to start the boarding procedure, independent of the timetable, if the number of persons in the queue is equal to the bus capacity
  - the bus driver will otherwise start the boarding procedure at the pre-scribed time, if there is at least one passenger queueing
  - to start the boarding procedure, the bus driver signals a number of passengers in the queue, up to the bus capacity, to enter the bus
  - when the last signaled passenger boards, she gives notice to the bus driver to start the transfer travel
  - upon arrival, the bus driver signals the passengers they may leave the bus
  - the last passenger exiting gives notice to the bus driver that everybody has already left.

Porter : passenger interaction

Two synchronization points are required

- *Arrival Lounge*
  - a synchronization point for the porter where he blocks while waiting for a plane to land, or for the end of daily activities
- *Baggage Collection Point*
  - an individual synchronization point for each passenger where she blocks while waiting for her bags.

Passenger : passenger interaction - 1

Two synchronization points are required

- *Arrival Terminal*
  - a common synchronization point for the passengers where they block while waiting for all fellow passengers traveling in the same plane to be ready to leave the airport, or to check in for the next leg of the journey
- *Departure Terminal*
  - a common synchronization point for the passengers where they block while waiting for all fellow passengers traveling in the same plane to be ready to leave the airport, or to check in for the next leg of the journey.

Passenger : passenger interaction - 2

These two synchronization points have to work in concert. For a given plane landing, the passengers are distributed between having the airport as their final destination and being in transit. Only when all of them reach the Arrival or the Departure Terminals, they may all proceed.

This means that there must be cross communication between this two shared regions: 1) for getting the number of passengers who are presently blocked, and 2) for wakening them up, when all of them have reached one of these regions.

Thus, the contents of the operations goHome and prepareNextLeg must be carefully written to avoid deadlock, because the calling of the operations associated with cross communication must be done outside the critical regions.

Bus driver : passenger interaction

Four synchronization points are required

- *Arrival Terminal Transfer Quay*
  - an individual synchronization point for each passenger where she blocks while queueing, or while the transfer travel takes place
  - a synchronization point for the bus driver where he blocks while waiting for the moment to signal the passengers to board the bus, or for the end of daily activities
  - a synchronization point for the bus driver where he blocks while waiting for all signaled passengers to enter the bus
- *Departure Terminal Transfer Quay*
  - a synchronization point for the bus driver where he blocks while waiting for all sat passengers to leave the bus.

Role of the General Repository of Information

The General Repository of Information works solely as the place where the visible internal state of the problem is stored. The *visible internal state* is defined by the set of variables whose value is printed in the logging file.

Whenever an entity (porter, passenger, bus driver) executes an operation that changes the values of some of these variables, the fact must be reported so that a new line group is printed in the logging file. The report operation must be *atomic*, that is, when two or more variables are changed, the report operation must be unique so that the new line group reflects all the changes that have taken place.

Code organization

The code should be organized in a hierarchical and logical way through the use of packages. At least, four packages should be considered

- *main* – containing the data types which define the simulation parameters and the application launcher
- *entities* – containing the data types which define the different entities and their states
- *shared regions* – containing the data types associated with regions of thread communication and synchronization
- *common infrastructures* – containing data types which implement different functionalities required to solve the problem.

The code should also be properly commented to produce useful documentation through the application of javadoc.

Code validation

Code validation should be carried out in two stages

- *stage 1* – the number of plane landings should be set to 1 and the logging file should be examined closely to assess the correctness of the algorithm
- *stage 2* – the number of plane landings should be set to the nominal value, 5, and, by using a shell script, instantiate a set of successive runs to check for racing conditions that were not detected before and that may lead to deadlock (at least, 100 successive runs should be considered).

```
for i in $(seq 1 10000)
do
     echo -e "\nRun n.º " $i
     java main.AirportConcSol
done
```