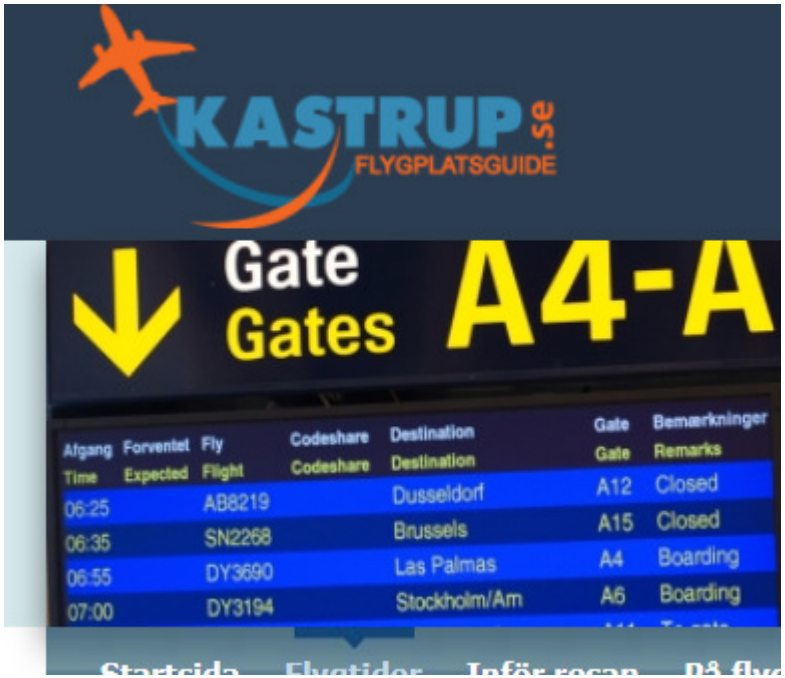
READERS/WRITERS – AN AIRLINE FLIGHT INFORMATION APPLICATION

1 OBJECTIVES

The main goals of this assignment are:

- Synchronization using semaphores
- Mutual exclusion using mutexes.
- Multiple readers and writers accessing a shared resource.



2 DESCRIPTION

There are many practical cases in which a Reader/Writer model can be implanted. In such scenarios, both the writers and readers are competing to access a shared resource.

In this assignment, your job is to simulate a practical case, a digital information board! The board can represent flight information in an airport, a board showing currency exchange rates, stock values, or train schedules. There are writer and reader threads that continuously want to access the board to change data and read data. The application should allow any number of concurrent reader threads, but the writer thread must have exclusive access to the buffer.

We pick-up the flight information board and make the following assumptions. Airline systems usually use a huge database for reservations and flight information, but here instead of a database, you can use a file with some test data (prepare it yourself). Simulate write and read actions using a Random object.

2.1 The flight information is monitored using a shared object **FlightInfoBoard** (hereafter referred to as **Board**).

2.2 For each flight, the **Board** maintains information about the Flight number, Departure time, Gate number and Status (Checking in, Boarding Departed, and Landed).

2.3 The **Board** contains a number of entries where each entry is used for a flight. The entries can be represented as an array or a list of strings.

2.4 Multiple readers and writers try to simultaneously access the **Board,** but only one writer is to be allowed to share the Board at a time and iff (if and only if) there is no reader using the **Board**.

2.5 Readers and writers should not be allowed to access the **Board** simultaneously.

2.6 Select a 12-hour table and randomize the access time for the writer threads to update less often than the reader threads.

3 SPECIFICATIONS AND REQUIREMENTS FOR A PASS GRADE (G)

3.1 Create an GUI-based application, design your GUI, and the way it should interact with the user. The GUI should show changes to the schedule.

Note: If you are using a "Controller" class, do not send the-reference to UI-class to the Controller methods; send the reference to the control on which your thread is operating.

3.2 Identify the classes you need to write. Consult your lab-supervisors if you are not sure. Write at least, a Writer class, a Reader class and a class for the shared resource in which you should also put your synchronization logics.

3.3 Use a **Readers/Writers** model and at least **two writers** and **three readers**, following the common protocol explained in above as well as during the lectures (see also lecture material).

3.4 Use semaphores and mutexes (binary semaphores in Java).

3.5 Test your application carefully before submitting.

NOTE: In case the above description is not sufficient, make your own assumptions. If you come across problems and uncertainties, solve them using your own judgement and decision.

4 GRADING AND SUBMISSION

Show your assignment to your lab supervisor during the scheduled hours in the labs, but before doing so, you must upload your work to Its' L. Make sure that you submit the correct version of your project and that you have compiled and tested your project before handing in. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code. It will not work on other computers. Projects that do not compile and run correctly, or is done with poor code quality, will be returned for completion and resubmission.

Compress all the files, folders and subfolders into a Zip, Rar or 7z file, and then upload it via the Assignment page on It's L. Click the button “Submit Answer” and attach your file. Do not send your project via mail!

Good Luck!

Farid Naisan,
Course Responsible and Instructor