COMP SCI 3004/7064 - Operating Systems Assignment 1

4 Aug, 2015

DUE: 9pm 7 Sept 2015

Important Notes

- Handins:
  - The deadline for submission of your assignment is **9pm the 7th of Sept, 2015**.
  - **For undergraduate students**, you may do this assignment as a team of two students and hand in one submission per team.
  - **For postgraduate students**, you have to do this assignment individually and make individual submissions.
  - Your program should be coded in **JAVA** and pass test runs on the three test files in "Assignments" of the course home page (`https://cs.adelaide.edu.au/users/third/os/15s2-os-adelaide`).
  - You need to use `svn` to upload and run your source code in the web submission system following "Web-submission instructions" stated at the end of this sheet. You should attach your and your partner's names and student numbers in your submission.
  - Late submissions will attract a penalty: the maximum mark you can obtain will be reduced by 25% per day (or part thereof) past the due date or any extension you are granted.
- Marking scheme:
  - 12 marks for dispatcher testing on 3 standard tests (4 marks per test).
  - 3 marks for the readability of your code and comments.

If you have any questions, please send them to the student discussion forum. This way you can all help each other and everyone gets to see the answers.

The assignment

The aim of this assignment is to improve your learning experience in the central content of process management on process synchronisation and scheduling algorithms. You are required to design an airline seat reservation system which allows multiple agents to reserve/cancel seats in a shared database, and coordinates them in such a way that at any time at most one agent enters the database. The database contains the classes of seats and their capacities in the concerned flight. In this assignment, we assume that seat classes and their capacities are first-class (10 seats), business-class (20 seats) and economy-class (70 seats).

In the reservation system, each agent request is executed by a process and all processes are maintained in a queue. Your reservation system schedules agent requests as follows:

- Processes arriving at different times are placed to the queue by *First-Come First-Served* (FCFS).

- Processes arriving at the same time are placed to the queue by *Highest Priority First* (HPF), where process priorities are arranged first by *process type* in the order of *cancellation—reservation*, then by *seat class* within the same type in the order of *first-class—business-class—economy-class*, and finally in the non-increasing order of the requested *number of seats* within the same class.

- *Queue jump*: To promptly reflect seat availability, there is a queue-jump mechanism that promotes any input *cancellation* process to the front of all *reservation* processes in the queue.

- For processes of same priority arriving at the same time, the system admits a maximum number of processes that can be accommodated by the current availability of that class if process type is *reservation*, and all the processes if process type is *cancellation.* The system places all un-admitted reservation processes at the end of the queue.

- For each batch of admitted processes, there is a process synchronisation mechanism that coordinates the processes to correctly ensure mutually exclusive access to the database. You must use a *semaphore* to implement this mechanism, and the implementation of the semaphore should avoid busy waiting by maintaining an off-line waitlist.

Note that processes once admitted will run non-preemptively until completion, and we assume that processing one seat's reservation/cancellation requires one time unit.

The output of your system must display the execution sequence of agent requests (processes) together with the value-changes of the semaphore during each phase of process synchronisation. The execution sequence must contain the content (agent IDs) of each admitted batch, each agent's execution of *wait* and *signal* operations and the resulting value of the semaphore after each operation, agent's entry to, operation (reserve/cancel) in and exit from the system, as illustrated in the following examples.

Examples

- Example 1:
  - Input:

Agent ID	Request Type	Class	Number of Seats	Arrival Time
1	R (Reservation)	F (First-class)	1	0
2	R	E (Economy-class)	10	0
3	R	E	2	0
4	R	E	2	0

  - Output:

```
Admit a batch (Agent 1)
Agent 1 executes wait operation, semaphore = 0
Agent 1 enters the system
Agent 1 reserves 1 seat in First-class
Agent 1 executes signal operation, semaphore = 1
Agent 1 exits the system
Admit a batch (Agent 2)
Agent 2 executes wait operation, semaphore = 0
Agent 2 enters the system
Agent 2 reserves 10 seats in Economy-class
Agent 2 executes signal operation, semaphore = 1
Agent 2 exits the system
Admit a batch (Agent 3, Agent 4)
```

```
Agent 3 executes wait operation, semaphore = 0
Agent 4 executes wait operation, semaphore = -1
Agent 3 enters the system
Agent 3 reserves 2 seats in Economy-class
Agent 3 executes signal operation, semaphore = 0
Agent 3 exits the system
Agent 4 enters the system
Agent 4 reserves 2 seats in Economy-class
Agent 4 executes signal operation, semaphore = 1
Agent 4 exits the system
```

Note that the execution order of processors within an admitted batch is not unique (depends on the implmentation of the waitlist). That is, in the above output, for the batch (Agent 3, Agent 4) Agent 4 may enter the system before Agent 3, resulting in an alternative execution order that is also correct.

- Example 2:
  - Input:

Agent ID	Request Type	Class	Number of Seats	Arrival Time
1	R	F	2	0
2	R	B (Business-class)	10	0
3	R	F	10	1
1	C (Cancellation)	F	1	4
2	C	B	2	14

  - Output:

```
Admit a batch (Agent 1)
Agent 1 executes wait operation, semaphore = 0
Agent 1 enters the system
Agent 1 reserves 2 seats in First-class
Agent 1 executes signal operation, semaphore = 1
Agent 1 exits the system
Admit a batch (Agent 2)
Agent 2 executes wait operation, semaphore = 0
Agent 2 enters the system
Agent 2 reserves 10 seats in Business-class
Agent 2 executes signal operation, semaphore = 1
Agent 2 exits the system
Admit a batch (Agent 1)
Agent 1 executes wait operation, semaphore = 0
Agent 1 enters the system
Agent 1 cancels 1 seat in First-class
Agent 1 executes signal operation, semaphore = 1
Agent 1 exits the system
Admit a batch (Agent 2)
Agent 2 executes wait operation, semaphore = 0
Agent 2 enters the system
Agent 2 cancels 2 seats in Business-class
Agent 2 executes signal operation, semaphore = 1
Agent 2 exits the system
Cannot admit a batch
Agent 3 cannot reserve any seats in this system
```

Web-submission instructions

- First, type the following command, all on one line (replacing xxxxxxx with your student ID):
  svn mkdir --parents -m "OS" https://version-control.adelaide.edu.au/svn/axxxxxxx/2015/s2/os/assignment1

- Then, check out this directory and add your files:
  ```
  svn co https://version-control.adelaide.edu.au/svn/axxxxxxx/2015/s2/os/assignment1
  cd assignment1
  svn add SchedulerDriver.java
  svn add StudentFile1.java
  svn add StudentFile2.java
  ...
  svn commit -m "assignment1 solution"
  ```

- Next, go to the web submission system at:
  `https://cs.adelaide.edu.au/services/websubmission/`
  Navigate to 2015, Semester 2, Operating Systems, Assignment 1. Then, click Tab "Make Submission" for this assignment and indicate that you agree to the declaration. The automark script will then check whether your code compiles. You can make as many resubmissions as you like. If your final solution does not compile you won't get any marks for this solution.

- We will test your codes as:
  ```
  javac SchedulerDriver.java
  java SchedulerDriver input.txt > output.txt
  ```

- The web-submission system for this assignment will be available at 6pm 4 Aug 2015.