COMP SCI 3004/7064 - Operating Systems Assignment 1

DUE: 9:00pm, 13th Sept, 2017

Important Notes

- Handins:
  - The deadline for submission of your assignment is **9pm the 13th of Sept, 2017**.
  - **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.
  - All implementations have to be done in **C++**.
  - You need to submit your source code using the web submission system. You should attach you and your partner's name and student number 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 online testing on 3 standard tests (4 marks per test).
  - 3 marks for the structure 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 process scheduling algorithms. You are required to design an online ticketing system for Coopers Stadium's public-seating section (red-coloured section in Coopers Stadium's seats map shown in Figure 1)

In the system, all the customers are grouped into five priority classes, ranged from 1 to 5, according to their loyalty (accumulated points) to this ticketing system. A larger priority number indicates a higher priority. All ticketing processes (purchase/cancellation) generated by a customer are assigned with the same priority number of that customer. To maximise the system service performance, you are required to implement a customer process scheduling algorithm using multi-level queue strategy with two queues: a high priority Queue 1 and a low priority Queue 2. Queue 1 has absolute priority over Queue 2. In other words, a customer process in Queue 2 will only be processed if there is no process in Queue 1. There is a *threshold* (=2) that is used to determine whether a process should remain in Queue 1 (priority $>$ *threshold*) or Queue 2 (priority $\leq$ *threshold*). Detailed actions in these two queues are listed below:

Queue 1: This is the high priority queue. All processes in this queue are treated in the way of combined *Highest Priority First* (HPF) and *Round Robin*. That is, select the highest priority process and process it for a ticket quota of 5 tickets (= 5 units of time quantum) non-preemptively, then move the process to the end of Queue 1. Processes of the same priority are

Figure 1: Stadium Map.

selected in their arrival order. The priority of a process in this queue is decreased by one every 5 runs of this process, i.e. when the process has processed 25 tickets under its current priority. If a process's priority goes below the *threshold* (=2), it is demoted from Queue 1 to Queue 2.

Queue 2: This is the low priority queue. Processes in this queue are handled in *Round Robin*. That is, select a process according to *First Come First Serve* and process it for a ticket quota of 20 tickets (= 20 time units) preemptively, then move the process to the end of Queue 2. Note: once a running process P in this queue is interrupted by a new arrival process in Queue 1, P will be preempted immediately even if it has not used up its time quantum. If the priority of a process in this queue reaches the *threshold* (=2) due to the Ageing mechanism below, it is promoted from Queue 2 to Queue 1.

Ageing mechanism: Because processes in Queue 2 will execute only when Queue 1 is empty, starvation may occur, i.e., some processes in Queue 2 may never get to run. To resolve this starvation issue, you must implement a mechanism which ages each process. This need not be done for every process run as it will slow the system down, but once every 9th run. That is, if a process has waited 8 runs (the interrupted process is counted as one run) of other processes since its last run, its priority number will be increased by one. In this way, the priority of each process in Queue 2 increases gradually in proportion to the waiting time since the last run.

Note: For the case if three processes of the same priority — a new arrival process A, a

preempted process B of Queue 1 (by Round-Robin) and a promoted process C from Queue 2 to Queue 1 — are put to the end of Queue 1 at the same time, their order will be A → B → C. Same rule is applied in Queue 2 regardless of process priority, i.e., new arrival first, preempted second and demoted last, if the three processes come to the end of Queue 2 at the same time, regardless of their priorities.

Test data format

Input

Each process is identified by a line in the input file (see "`input-sample.txt`" in the assignment folder). The line describes the process ID, arrival time, priority, age and the total tickets required. For example s1 3 1 0 50 describes a process s1 which arrived at time 3 with priority 1 and age 0, and requires 50 tickets. One ticket processing consumes one time unit.

Output

The output provides information of each process execution. The line starts with the process ID, arrival and termination times, ready time (the first time the system processes the process) and durations of running and waiting (see "`output-sample.txt`" in the assignment folder).

You may monitor the execution of your code by displaying all its intermediate outputs as shown in the sample "`detailed-output.txt`" in the assignment folder. Note that this is merely for your own code debugging purpose, and should not be presented in the final output.

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/2017/s2/os/assignment1
  ```
- Then, check out this directory and add your files:
  ```
  svn co https://version-control.adelaide.edu.au/svn/axxxxxxx/2017/s2/os/assignment1
  cd assignment1
  svn add TicketBooker.cpp
  svn add StudentFile1.cpp
  svn add StudentFile2.cpp
  ...
  svn commit -m "assignment1 solution"
  ```
- Next, go to the web submission system at:
  `https://cs.adelaide.edu.au/services/websubmission/`
  Navigate to 2017, 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 code does not compile you won't get any marks for the online testing part.
- We will test your codes by the following Linux commands:
  ```
  g++ TicketBooker.cpp -o TicketBooker
  ./TicketBooker input.txt > output.txt
  ```