COMP SCI 3004/7064 - Operating Systems Assignment 1

DUE: 16 September 2020 at 11:30 PM

Important Notes

- Handins:
  - The deadline for submission of your assignment is **11:30 PM 16 September 2020**.
  - **For undergraduate students**, you may do this assignment as a team of two students and upload only **one submission per team**. You should clearly state the student IDs and names of both members in File Header comments of your submission file.
  - **For postgraduate students**, you have to do this assignment **individually** and make individual submissions with your name and student ID included in File Header comments of your submission file.
  - All implementations must be done in **C++**.
  - You need to submit your source code using School's web submission system following the instructions given at the end of this assignment sheet.
  - 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:
  - 20 marks for 10 randomly generated tests (2 marks per test).

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 on process scheduling algorithms. You are required to design an online ticketing system for the Melbourne Arena, one of the largest stadiums in Australia that offers state-of-the-art facility for hosting sports and entertainment events including the Australian Open tennis tournament held annually. Specifically, there are 10,500 seats for public booking via this system for a single match. Figure 1 shows the distribution of the seat sections, where all the yellow and green areas are for public reservations.

In the system, all the customers are grouped into five priority classes (numbers), ranged from 1 to 5, according to their loyalty (accumulated points) to this ticketing system. A smaller 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 scheduling algorithm using multi-level queue strategy with two queues: a high priority Queue 1 and a low priority Queue 2, where Queue 1 has absolute priority over Queue 2. In other words, customers in Queue 2 will only be processed if there is no customer in Queue 1.

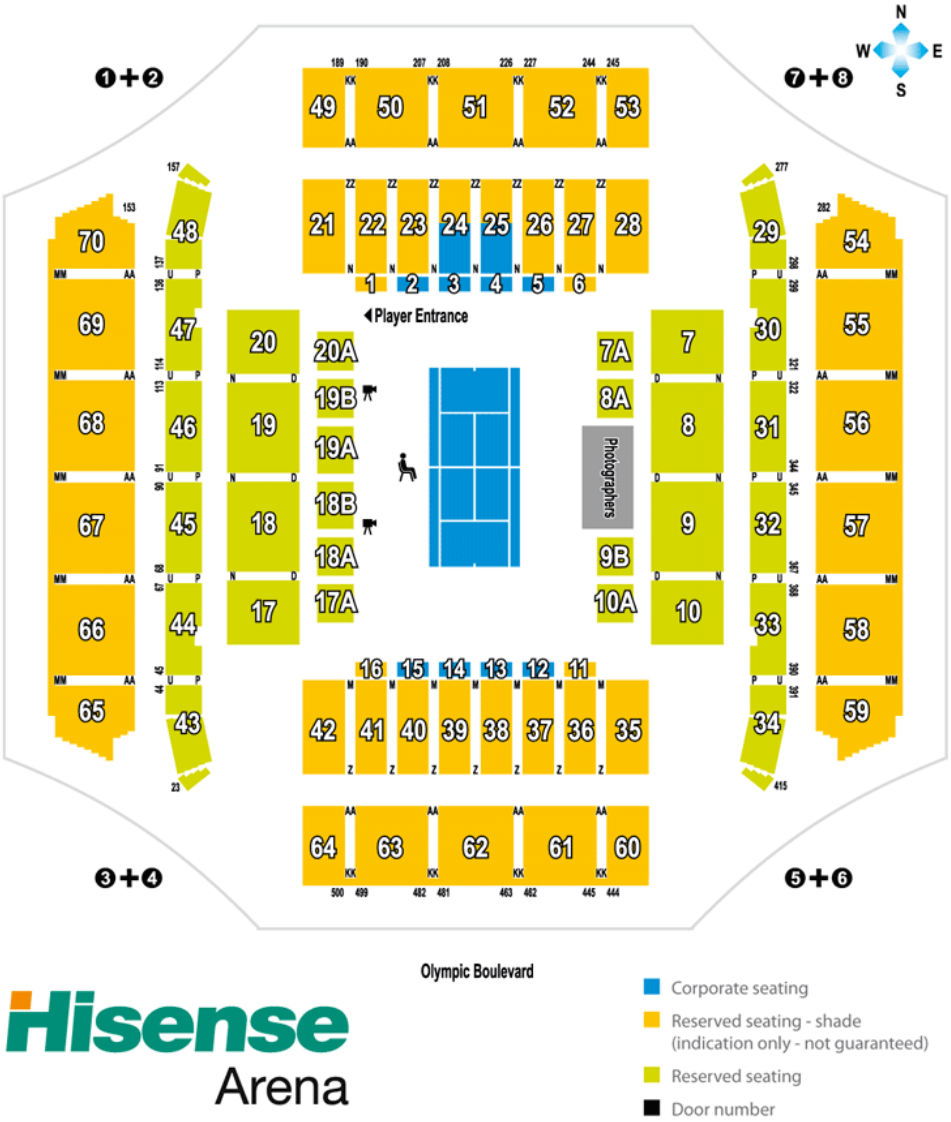

Figure 1: Hisense Arena

A new customer on arrival will be put on Queue 1 if his/her priority number ≤ 3 or Queue 2 otherwise.

Detailed actions in these queues are described below:

Queue 1: This is the High-Priority queue (priority number ≤ 3). Customers in this queue are treated in the way of combined *Highest Priority First* (HPF) and *Weighted Round Robin* (WRR — definition see below) on priority as follows: Select the highest priority customer (customers with the same priority are processed in their arrival order), and then process this customer's request for a ticket quota of $N = \frac{weighted\_time\_quantum}{5\ time\ units}$ tickets non-preemptively, then move this customer to the end of (the priority-subqueue of) Queue 1, where

- $weighted\_time\_quantum = (10 - \text{customer's priority number}) \times 10$
- 1 ticket costs 5 time units to process.
- customer's priority number is the customer's current priority number.

Weighted Round Robin (WRR): Given n processes $P_1, P_2, \ldots, P_n$, where process P_i has a weight w_i $(1 \leq i \leq n)$, WRR allocates P_i a weighted time quantum of w_i time units (i.e.

a share of $\frac{w_i}{\sum_{i=1}^{n} w_i}$ CPU time). In this assignment, to simplify the implementation, a process' weight is (10 − customer's priority number), and customer's priority number is the customer's current priority number.

Customers of the same priority are processed in their arrival order. The priority of a customer in this queue is decreased by 1 every 2 runs of this customer, i.e. when a customer has been processed 2 times under its current priority, its priority number will be increased by 1. When a customer's priority number reaches 4, it will be demoted from Queue 1 to Queue 2.

For example, the priority number 2 in this queue is decreased by 1 after 2 runs of this customer, i.e. once booked $2 \times N = 2 \times \frac{(10-2)\times 10}{5} = 32$ tickets its priority will become 3 and its ticket quota for the next 2 runs will be $N = \frac{(10-3)\times 10}{5} = 14$ (instead of 16 in its previous 2 runs).

Queue 2: This is the Low-Priority queue (priority number ≥ 4). Customers in this queue are handled in *Shortest Remaining Time First*. That is, select the customer with *shortest remaining time*, regardless of their priorities, and process this customer's request preemptively (to any a new arrival or promoted customer in Queue 1).

Customers of the same job length (time units) are processed in their priority number increasing order. Customers of the same job length and priority are processed in their arrival order.

Note: once a running customer X in this queue is interrupted by a promoted or new arriving customer in Queue 1, customer X will quit execution immediately and the new customer in Queue 1 will get the CPU to run.

Aging mechanism

Because Queue 1 has priority over Queue 2, and the HPF strategy is applied, starvation may occur. That is, some customers in Queue 2 may never get to run because there is always a job with higher priority in Queue 1. To address the starvation issue, you must implement the following aging mechanism that dynamically promotes a customer of Queue 2 to Queue 1 when he/she experiences starvation.

The priority of a customer in this queue (Queue 2) is increased by one every 100 time units run of other customers since the last run of this customer. That is, if a customer has waited 100 time units of other customers since his/her last run, his/her priority number will be decreased by one. In this way, the priority number of each customer in Queue 2 decreases gradually in proportion to the waiting time since the last run. When a customer's priority number reaches 3, it will be promoted from Queue 2 to Queue 1.

Note:

- **Order of simultaneous arrivals of same-priority customers:**

For customers arriving at the same time with the same priority to both queues, we assume their arrival order follows the increasing order of their customer IDs.

- **Order of new arrivals, preemption and promotion**

- For Queue 1, there may be three types of customers with the same priority arriving simultaneously at the end of (the priority-subqueue of) Queue 1 — a new arrival customer A to Queue 1, a customer B with this priority of Queue 1 moved to the end of Queue 1 (by Weighted-Round-Robin), and a promoted customer C from Queue 2 to Queue 1. In this case, their execution) order in Queue 1 will be $Queue1 \rightarrow A \rightarrow B \rightarrow C$

- For Queue 2, there may be two types of customers arriving simultaneously to Queue 2 — a new arrival customer A to Queue 2 and a demoted customer B from Queue 1, their relative arrival order in Queue 2 will be $Queue2 \rightarrow B \rightarrow A$.

Test

Input

We have N customers, Each customer is identified by a line in the input file. The line describes the customer ID, arrival time(Can be divisible by 5), priority, age and the total tickets required. For example a1 5 1 0 50 describes a customer a1 which arrived at time 5 with priority number 1 and age 0, and requires 50 tickets. One ticket processing consumes 5 time unit.

Output

The output includes N+1 lines. it provides information of each customer execution. First line is string include "name arrival end ready running waiting". The next N lines (sorted by termination time) should report the customer ID, arrival and termination times, ready time (the first time the system processes his/her request) and durations of running and waiting.

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/2020/s2/os/assignment1
- Then, check out this directory and add your files:
  svn co https://version-control.adelaide.edu.au/svn/axxxxxxx/2020/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 2020, 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 by the following Linux commands:
  g++ TicketBooker.cpp -o TicketBooker
  ./TicketBooker input.txt>output.txt
- Do not forget to include your name(s) and student ID(s) in File Header comments of TicketBooker.cpp