Programming Assignment 3

CSCD300 Data Structure

Instructor: Dr. Bojian Xu
Eastern Washington University, Cheney, Washington

Due: 11:59pm, July 8, 2014 (Tuesday)

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own work, without referring to anybody else's solution.
2. No one should give his/her code to anyone else.
3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Blackboard system). Send in whatever you have by the deadline.
4. Every source code file must have the author's name on the top.
5. All source code should be commented reasonably well.
6. Sharing any content of this assignment and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Project: Implementing the round-robin process scheduling

Roughly speaking from the operating system's point of view, a *process* is a running program. Modern computer operating systems, such as Microsoft Windows, various Linux deliveries, and MacOS, support multi-process (or also called multi-task or multi-job) computing, meaning the user can run multiple programs simultaneously on one machine. However, any single computer processing unit (such as a core within a multi-core CPU or a single-core CPU) can do the computation for only one process at any moment. So, how to run the multiple processes "simultaneously" on one machine ? The way that the operating system does is to let the CPU serve one process for a very short period of time and then switch the CPU to serve another process and so on. By doing so, the user feels that all the processes are running in parallel. This raises the following important and challenging question: how to schedule/queue the processes so that they can be served "fairly" and the overall multi-process system looks working quite smoothly ? This question is about process scheduling and there are many different scheduling policies/strategies/algorithms dealing with this issue by considering different needs and targeting different goals. In this programming assignment, we want to implement a very basic and simple version of the *Round-Robin* [1] scheduling by using the circular linked list data structure.

[1] `http://en.wikipedia.org/wiki/Round-robin_scheduling`

Round-robin scheduling. In the round-robin scheduling, all processes are organized in a circle. The CPU serves the processes one by one by going around the circle. Every time a process is served, it will be served for a pre-determined fixed amount of time unless the process terminates earlier. For example, if the following are the processes currently running in a system, where each process is represented by a ⟨process id, process's CPU time needs⟩ tuple:

$$\langle 1, 10\rangle, \langle 2, 7\rangle, \langle 3, 19\rangle, \langle 4, 12\rangle, \langle 5, 16\rangle$$

Suppose the length of each CPU service time is 5 and all the processes are served in the order as they are shown above. Then we will have the following processes along with their CPU time needs after each round of CPU service:

- After the first round of service,: $\langle 1, 5\rangle, \langle 2, 2\rangle, \langle 3, 14\rangle, \langle 4, 7\rangle, \langle 5, 11\rangle$
- After the second round of service: $\langle 3, 9\rangle, \langle 4, 2\rangle, \langle 5, 6\rangle$
- After the third round of service: $\langle 3, 4\rangle, \langle 5, 1\rangle$
- After the fourth round of service, all processes terminate.

Specification of your program

1. Your program should be named as: `Test_RoundRobin.java`
2. The input and output of your program.

   There are two inputs to your program. One is a text file, that contains a list of ⟨process id, process's CPU time needs⟩ tuples. Each line of the text file is one tuple. The two fileds of each tuple is separated by the comma symbol. **You can assume all process ids are distinct.** The file name should be supplied to your program as a command line parameter. The other input, which is also supplied as a parameter to your program, is a positive integer number representing the length of each CPU service.

   The output should be a sequence of process ids that are in the ascending order of their termination time, separated by the comma symbol. **We require all processes be served at the order of their ids in each round.**

Implementation of your program

You need to construct a circular singly linked list to organize all the processes from the input file, where each node represents a process. **The nodes in the circular singly linked list are in the ascending order of process ids**, so that in each round of service, the processes will be served at the ascending order of their ids. You program will then scan the circular linked list round by round starting from the process with the smallest id. Every time after a process is served for the specified amount of time, if the process has no more CPU time needs meaning the process has terminated, then the linked list node representing that process should be discarded from the linked list and the corresponding process id should be printed on the screen.

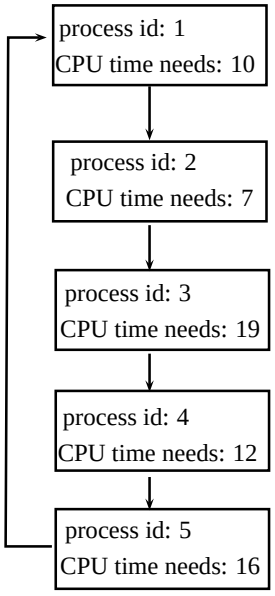
An example

Suppose we supply to your program with the file named `data.txt` that has the following content, along with each CPU service time length being 5,

```
1,10
5,16
2,7
4,12
3,19
```

Then, the command line you should type would be:
`$java Test_RoundRobin data.txt 5`

Then your program will construct the following circular singly linked list to start with:



and the output should be:
`$1,2,4,3,5`

Submission

- All your work files must be saved in one folder, named: **firstname_lastname_EWUID_cscd300_prog3**
  (1) We use the underline '_' not the dash '-'.
  (2) All letters are in the lower case including your name's initial letters.
  (3) If you have middle name(s), you don't have to put them into the submission's filename.
  (4) If your name contains the dash symbol '-', you can keep them.
- You need to include a pure ascii text file in the above folder, which contains the description of your implementation of the circular linked list data structure for the round-robin scheduling.
- You then compress the above whole folder into a .zip file.
- Submit .zip file onto the Blackboard system by the deadline.