

## LAB 1: Multi-Level Feedback Queue Scheduler

### Objectives:

1. Understand CPU scheduler
2. Implement an algorithm
3. Empirical analysis of an algorithm
4. Gain experience with an industry standard OS

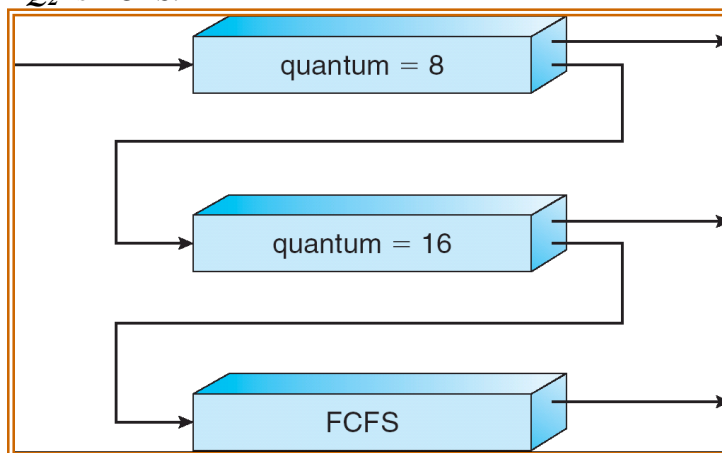
**Due:** Wednesday, October 8, 2008. 11:59:59 PM (EST)

These will be individual projects. The main purpose of this Lab assignment, i.e. LAB1, is to understand and implement a Multi-level feedback queue CPU scheduler.

For LAB1, you may choose any programming language, such as .NET, Visual C++, Java, C, etc. The students are encouraged to choose the programming language that supports GUI, and runs on Windows. Also, the other OS that support GUI desktop, e.g. Gnome and KDE for Linux, are possible. Finally, your final program should be compiled and run on GA's computer. To do that, you need to provide all materials as well as source code, such as run time module, integrated development environment (Visual Studio, .NET or C#), and any other program to run your program.

You are to implement a multi-level feedback queue scheduler  $Q$ . Your implementation of  $Q$  should satisfy the following requirements:

- The scheduler  $Q$  consists of 3 linear queues, i.e.,  $Q_0$ ,  $Q_1$ , and  $Q_2$ .
  - $Q_0$  is round robin with time quantum 8 ( $RR_8$ ),
  - $Q_1$  is round robin with time quantum 16 ( $RR_{16}$ ), and
  - $Q_2$  is  $FCFS$ .



- Process cannot be executed in the lower queue if there are any jobs in all higher queues. For example,  $Q_0$  has 5 processes,  $Q_1$  has 1 process, and  $Q_2$  has 1 process. Then, first the process in  $Q_2$  should be executed (and completed), and then a

process in  $Q_1$  is executed and move to  $Q_2$  (then executed and completed). Finally,  $Q_0$  will get CPU resource.

- A new process enters queue  $Q_0$  which is served  $RR_8$ .
- When it gains CPU, a process receives 8 milliseconds.
- If it does not finish in 8 milliseconds, process is moved to queue  $Q_1$ .
- At  $Q_1$  process is again served  $RR_{16}$  and receives 16 additional milliseconds.
- If it still does not complete, it is preempted and moved to queue  $Q_2$ .
- The processes are to be created with the following fields in the PCB (Process Control Block): Process ID, arrival time, actual execution time, queue number. The creation is done randomly, and includes at least 20 processes.
- Output should include a time line, i.e., every time step, indicate which processes are created (if any), which ones are completed (if any), processes which moved into different queue, etc. For example,

Time	Description
0 ms	P0 is created. P0 enters Q0. P0 is selected.
3 ms	P1 is created. P1 enters Q0.
8 ms	P0 is aged. P0 moves to Q1. P0 is selected.
...	...

Also, you can implement GUI displaying each queue status including queue, current process, and time (Optional).

- Context switching time is ignored in this project.

After completing the implementation and doing a few sample runs, compute the following performance measure.

- Average waiting time.
- Average turnaround time.
- Average response time.

### Write-up

You should submit a write-up as well as your program. Your write-up should include analysis of performance measure with various parameter values, any known bugs, limitations, and assumptions in your program. This write-up should be in text-format and titled as 'README'. It should be submitted along with your code. GA will use the

'README' file to compile (or install) and run your program. If the GA has trouble with your program then he will contact you to makeup it.

## **Demonstration**

After the project is submitted, each student should demonstrate it in front of instructor and GA. The sign-up sheet will be provided before due date, and the student can choose 15 minutes time slot for the demonstration.

## **Submission**

You will submit your program using blackboard digital drop-box (<http://bb.ctdlc.org/>). If you have any trouble to use blackboard, you can contact GA or instructor.

Make the Name "CPSC408: LAB1".

You should zip your source files and write-up (README) into a single file and submit it. Be sure that you include everything necessary to unzip this file on another machine and compile and run it. This might includes forms, modules, classes, configuration file, etc. DO NOT include the executable file itself, we will recompile it. The name of your submitted zip file should be your "UB account"\_"ID"\_"lab1". For example, "jelee\_000000\_lab1.zip".

Make sure your name and UB ID are listed in both your write-up and source code. You may resubmit your program at any time. However, please do not ask the GA to grade your program and then resubmit it. The submission time of the latest submission will be used for determining whether the assignment is on time or not. Late submissions will be accepted at a penalty of 10 points per day. In other words, it may pay you to do this project early on the off chance that something prohibits your submitting it in a timely way. If your program is not working by the deadline, submit it anyway and review it together with GA for partial credit. Do not take a zero on any lab just because the program isn't working yet. If you have trouble getting started, ask the professor or the GA.

## **Grading**

- 30 Accuracy of RR
- 20 Accuracy of FCFS
- 20 Feed back with aging
- 20 Analysis including performance measurement
- 05 Write-up
- 05 Comments in code + Write-up
- Bonus +10 for GUI

To receive full credit for comments in the code you should have headers at the start of every module, subroutine, or function explaining the inputs, outputs and function of the

module. You should have a comment on every data item explaining what it is about. (Almost) every line of code should have a comment explaining what is going on. A comment such as `/* add 1 to counter */` will not be sufficient. The comment should explain what is being counted.