GEORGE MASON UNIVERSITY

Department of Computer Science

CS 471 - Operating Systems, Spring 2015 Project #1
Due: February 24, 11:59 PM

---

1 Overview

In this project, you will implement **a simulator for process scheduling on a multi-programmed system with one CPU**. Your program will read the characteristics of the workload from an input file and then will simulate the execution of the processes. The simulator should print information about the occurrence of important "events" (such as process dispatch or completion points) and the status of various system resources after each event to an output file. At the end of the simulation, your program will also print **the average waiting time** and **the average turnaround time** per process.

Below we provide the specifications and details about the project as well as some suggestions. **Please read all the parts of this hand-out carefully before starting to work on your project**.

2 CPU scheduling algorithm

The CPU Scheduling algorithm that you will implement will be a **Multi-level Feedback (MLF) Queue** with **3 levels/queues**, similar to the one we discussed on Slide 3.29. The project involves a generalization of that structure to multiple CPU and I/O bursts. One difference is that instead of the First-Come-First-Served algorithm, preemptive Shortest-Remaining-Time-First (SRTF) algorithm will be used in the third-level queue.

We will use the symbols Q_1, Q_2 and Q_3 to denote the three queues in the system. At any time, processes in Q_i will have strictly higher priority than processes in Q_{i+1}. In Q_1 and Q_2, the Round-Robin (RR) scheduling policy will be used, with corresponding time quantums tq_1 and tq_2 ($tq_2 > tq_1$). In Q_3, the preemptive Shortest-Remaining-Time-First (SRTF) algorithm will be used. At the beginning of each CPU burst (or, at each arrival time), the process will be put to (the end of) Q_1 and will receive service according to the rules of RR algorithm with time quantum tq_1. After receiving tq_1 units of service, if its CPU burst is not completed, it will be moved (demoted) to Q_2. Similarly, it will be moved from Q_2 to Q_3 if its CPU burst continues after receiving an additional tq_2 units of CPU time.

The algorithm is fully preemptive: for example, if a new process arrives to (or, rejoins) Q_1 while a process P in Q_2 is executing on the CPU, P will be preempted. In that case, P will use the remaining of its time quantum when Q_1 becomes empty again. In a similar way, processes in Q_3 may be preempted numerous times by processes in Q_1. Also note that, a process demoted to Q_3 may complete its CPU burst earlier or later than another process already in Q_3, depending on the relative lengths of the CPU bursts, as will be decided by the SRTF algorithm used in that queue.

It is important to remember that when a new CPU burst of P starts at the completion of its I/O burst, P will be put again to the highest-level queue Q_1 and begin to execute with full time quantum tq_1 regardless of its execution history during its previous CPU bursts.

3 I/O Device Operation

There may be multiple I/O devices in the system. A given I/O device may be used by at most one process at a time. At the completion of a CPU burst, the process will request its next I/O on a specific device, and it will join the *Device Queue* associated with that device.

An I/O device can be either in *idle* or *busy* state at a given time. Processes on a Device Queue will be served for I/O on that device on **First-Come-First-Served** basis (even though CPU scheduling is performed through the MLF Queue). At any time, the process at the head of a Device Queue is assumed to be performing the I/O operation on that device. Note that due to the nature of I/O operations, multiple I/O devices may be serving different processes in parallel, while the CPU is executing a completely different process.

4 Basic functions of the simulator

The input file will provide values of the time quantums as well as the properties of individual processes. It will have the following generic format:

Time Quantum 1:
Time Quantum 2:

Process ID:
Arrival time:
CPU burst:
I/O burst:
I/O device id:
CPU burst:
I/O burst:
I/O device id:
....
Process ID:
Arrival time:
CPU burst:
I/O burst:
I/O device id:
CPU burst:
I/O burst:
I/O device id:
....

Your program will simulate the execution from time = 0 until all the processes have completed all the CPU and I/O bursts. The fields Time Quantum 1 and Time Quantum 2 will indicate the time quantum (slice) values tq_1 and tq_2 for Queue1 and Queue2, respectively. The information about each process will be given in the input file by its identity (an integer) denoted by Process Id, the time it is submitted to the system (Arrival time) and an alternating sequence of CPU bursts and I/O bursts. The integer field CPU burst indicates the duration of the CPU burst, while the integer in the field I/O burst gives the duration of the I/O burst. The field I/O device id gives the id of the I/O device on which the involved I/O takes place. As an example, suppose that a specific process has the following characteristics:

Process ID: 2
Arrival Time: 15
CPU burst: 20
I/O burst: 100
I/O device id: 2
CPU burst: 30
I/O burst: 150
I/O device id: 1

This means that Process 2 arrives to the system at time = 15, starts with a CPU burst of 20 time units, and then undergoes an I/O burst for 100 units on I/O Device #2. When this I/O operation is completed, it has another CPU burst for 30 units. It completes after doing a second I/O for 150 time units, this time on I/O Device #1. Thus, it would complete at time = 315 if it were the only process in the system. However, other competing processes may force it to wait for the CPU and/or I/O devices, effectively increasing its completion time.

Your simulator must generate an output file that explicitly shows the trace of important "events", including:

- Process arrival
- CPU request (start of the CPU burst: when a process joins Q_1)
- Process dispatch (when a process is moved to running state)
- Process preemption
- Process demotion to a lower level queue
- CPU burst completion
- Process completion
- I/O device request (when a process is moved to a device queue)
- I/O start
- I/O burst completion

Whenever there is a change in any of ready queues or any of the device queues, your program must also display the updated contents of the corresponding queue(s). Your output files should also contain information about process moves from Q_i to Q_{i+1}, since your project involves a Multi-level Feedback Queue. The last two entries in the output file must be the average waiting time and the average turnaround time of the processes.

Two input/output file pairs are provided in the Blackboard CS 471 Project 1 folder for your convenience. The first pair is for a (simpler) system with only one ready queue and Round-Robin policy; the second pair is for a 3-level MLF queue (the subject of this project).

You can assume that the input file will contain information about at most 10 (ten) processes, numbered from 1 to 10. Each process will have at most 10 CPU and I/O bursts. There are at most 5 I/O devices in the system, again numbered from 1 to 5.

5 Implementation

In a simple implementation, you can keep an integer (or long) variable representing your "virtual clock". Then you can increment its value until all the processes complete their executions, taking appropriate CPU and device scheduling actions along the way as processes arrive and complete.

Your system will be based on *multiprogramming* principle. Thus, the CPU scheduler must be invoked to select the next process to run from the Multi-level Feedback Queue when the currently running process completes, is blocked for I/O, completes its I/O, uses its entire time quantum (in Q_1 or Q_2), or is demoted to the next level. Carefully review the lecture slides, related textbook sections and your notes to see when the CPU scheduler is invoked. In case that multiple processes have the same priority (for example, two processes may arrive at the same time), ties can be broken arbitrarily.

Each process will be represented by a **Process Control Block (PCB)**, having separate fields for at least *Process Id, Arrival time* and *State*[1]. You may incorporate additional fields to the PCB if it results in a more efficient implementation. You will also maintain **ready queues** associated with each of three levels; these should be data structures (linked lists) involving the PCBs of the ready processes.

The I/O devices can be represented by **Device Descriptors (DDs)** having the fields for the *state* and possibly a *pointer/link* to the corresponding Device Queues. A Device Queue will be a linked list of the PCBs, with that of the process currently performing I/O, and those waiting for I/O, on that device.

Since the focus of this project is on process scheduling, you can assume that the main memory is large enough to accommodate all the processes at all times.

IMPORTANT: In this project, *you cannot use arrays*, except for string manipulation and file reading/writing purposes. Your scheduling related data structures should be based on pointers and structs.

[1] As we discussed in class, in real systems, a PCB has many other fields describing various attributes of the processes; however, a simple implementation is sufficient for this project.

6 Operational Details

You will use C programming language in this project. Your program *must* compile and run on either mason.gmu.edu or zeus.vse.gmu.edu – it will be tested and graded on these systems. If you develop your program on another system, you should complete porting and testing tasks on mason or zeus before the submission deadline.

Please do follow the format of the input file as given above: during the grading process we will use our own input files in that format to test your program. Your program should skip any (sequence of) blank lines in the input file. If your program works with input files with different formats, but not with the one given above, a penalty will apply.

In this project, you can work alone or with another CS 471 student. No project team can have more than two members. Both members of a team will get the same project grade and there will not be penalty or bonus points for those working alone. GMU Honor Code will be enforced by comparing the source codes, possibly by automated mechanisms. You are not allowed to co-operate with any other person except your project partner (if you have one).

7 Submission Details

Submissions will have both soft and hard copy components.

The soft copy components will be submitted through *Blackboard*. You will need to submit all the softcopy components (listed below) as a single compressed *zip* or *tar* file. Submit your compressed file through the Blackboard by February 24, 11:59 PM. Before submitting your compressed file through the Blackboard, you should rename it to reflect the project number and your name(s). For example, if Mike Smith and Amy White are jointly submitting a tar file, their submission file should be named as: project1-m-smith-a-white.tar. On Blackboard, select the Projects link, then the Project 1 folder. In addition to the assignment specification file, you will see a link to submit your compressed tar or zip file. *Please do not use Windows-based compression programs such as WinZip or WinRar* – use tools such as zip, gzip, tar or compress (on zeus or mason).

The soft copy components that need to be submitted in a single *tar* or *zip* file are as follows:

- a README file with:
  - information on whether you prefer that your program be tested on *mason* or *zeus*
  - instructions on how to compile and run your program
  - a statement summarizing the known problems in your implementation (if any).
- the source code(s) of your program(s). Your source file(s) should include in the first commented lines your name(s) and G number(s).
- a write-up explaining the high-level design of your program and describing in detail the data structures you used for the MLF Queues and Device Queues.

You should also submit the hard copy of all the components above to the instructor at the beginning of the class meeting on February 25, 4:30 PM. Failure to follow the submission guidelines may result in penalties.

Late penalty for Project 1 will be 25% per day. Submissions that are late by four days (or more) will not be accepted.

8 Suggestions

- **Start early!** Postponing the project to the very last days will probably result in an incomplete or late submission.
- Assuming that coordination is feasible for you, it is suggested that you form a team with another CS 471 student. In this way, you can make design decisions together and share the implementation responsibility.
- Choose your partner wisely. Efficient cooperation with your partner can provide a better project with less effort only if you are well-coordinated. If you have a partner go over the project specification with him/her as soon as possible and decide on the individual responsibilities.
- Carefully review this specification, your lecture notes and the textbook before starting to code to make sure that *you* thoroughly understand the system execution mechanism and the rules of the specific scheduling algorithms. Otherwise, you may have an incorrect implementation. It may be also helpful to examine the example trace files provided on the project web page.
- You can direct your programming and system questions to CS 471 TA Rahul Goswami (*rgoswam2@masonlive.gmu.edu*). However, your questions should be *project-specific*: since C programming is covered in your previous courses, you can re-visit your programming language textbooks and manuals for general questions about programming. Conceptual questions about the project should be directed to the instructor (*aydin@cs.gmu.edu*). Please add the prefix **CS 471: Project 1** to the subject line in any correspondence with the TA or the instructor.