CPSC 3500
Programming Assignment: CPU Scheduling
Due: April 20, 11:59 PM (on Canvas)

NOTE: Please submit your work by April 20 on Canvas. This assignment is to be done individually; you can discuss the problem with your classmates, but you should code your own solutions independently. This programming assignment would be 20% of the total assignment grade across all homeworks.

In this assignment you will implement and simulate the performance of some of the CPU scheduling algorithms that we have studied in class. You can either implement the following tasks as a single C++ program, or you can split them. Make your choice.

Directions:

1. Irrespective of any design decisions you make, your program should accept, as command line input, the accompanying "jobs.txt" file.
2. Use the provided "jobs.txt" file for answering the questions that follow each scheduling algorithm. This file has three columns (comma separated): the first column contains the job Id, the second column indicates the CPU burst, and column three is the arrival time of the corresponding job.
3. For final evaluation of your code, **we will be using a different input file**, which will be in exactly the same format as "jobs.txt," but will have different job parameters (such as different job lengths, different number of jobs, etc.). For full credit, your code should work on different job input files.
4. Your programs will primarily be evaluated for functionality. The greater the number of test cases they pass, the greater will be the score.
5. Partial credit can be given. However, a program failing several test cases will receive a very low grade.
6. **No changes to the submitted programs** (however minor) will be accepted after the submission. Please test your programs thoroughly before submission and ensure that you are submitting the best version.
7. **Your programs will be evaluated on the CS1 server.** Please ensure they compile and run as expected on the CS1 server. Programs that fail to compile or run on the CS1 server will receive a zero.
8. Testing and debugging are integral part of programming. Before submission, you are responsible for creating test cases and testing your programs.
9. Sharing of code in any form is strictly prohibited. You can share the test cases, though.

---

1. To begin, we will be implementing the **Multiqueue Adaptive Scheduling** (MAS) algorithm. The MAS scheduling algorithm has three queues, namely q0, q1, q2. MAS uses a preemptive priority scheduling among the queues:

- Queue q0 has the highest priority followed by q1, followed by q2.
- The scheduler will first execute all the processes in q0. Only when q0 is empty will it execute processes in q1. Similarly, processes in q2 will be executed only if q0 and q1 are empty.
- A process that arrives in q1 will preempt an executing process from q2. Similarly, an executing process in q1 will be preempted by an incoming process in q0.

Each of the three queues use different scheduling policy:

- An incoming (new) process is always added at the end of q0. A process in q0 is allowed a time quantum of 6 milliseconds (ms). If it does not finish in 6 ms, it is moved to the tail of q1. Each process in q0 is scheduled for 6 ms on the CPU in FCFS order.
- When q0 is empty, the processes in q1 are scheduled on the CPU in FCFS order. Each process is allowed a time quantum of 12 ms. If a process does not finish in 12 ms, it is moved to the tail of q2.
- Processes in q2 are run using the FCFS scheduling algorithm but are run only when q0 and q1 are empty (as summarized above).

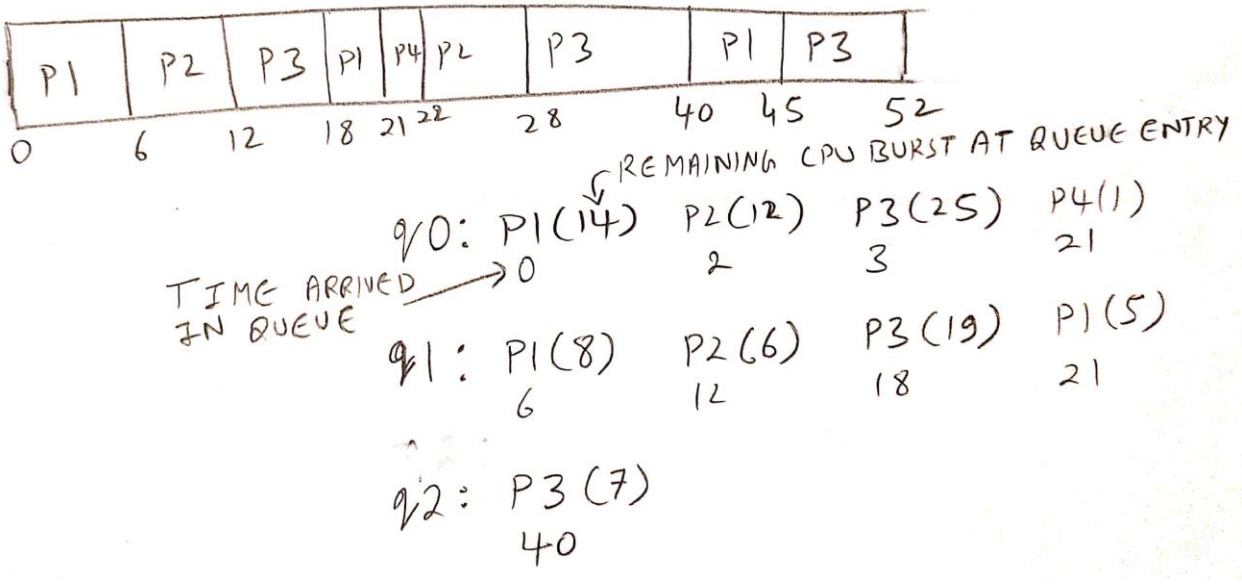
Any time there is a tie among processes for executing next, use the process Id to break the tie (process with smaller Id goes first). If a process executing in q1 or q2 is preempted by an incoming process in the higher priority queues, it would be added at the tail of its current queue. E.g., if a process (P12) belonging to q2 is preempted by a new arrival in q1, the preempted process (P12) will be added at the end (tail) of q2.

Use your implementation of the MAS algorithm to compute the following for processes in the supplied "jobs.txt". (When executed, your program should print the following pieces of information on screen for the MAS algorithm.) (**55 points**)

a) Average **turnaround time** for all the jobs.
b) Average **waiting time** for all the jobs.
c) Besides, report the order in which processes were allowed to execute on the CPU along with the process' final termination time. For instance, for the following set of processes scheduled using MAS, your program should print: 1(45), 2(28), 3(52), 1(45), 4(22), 2(28), 3(52), 1(45), 3(52)

Process	Arrival Time	Burst Time
P_1	0	14
P_2	2	12
P_3	3	25
P_4	21	1

Below is the expected schedule for the MAS algorithm.

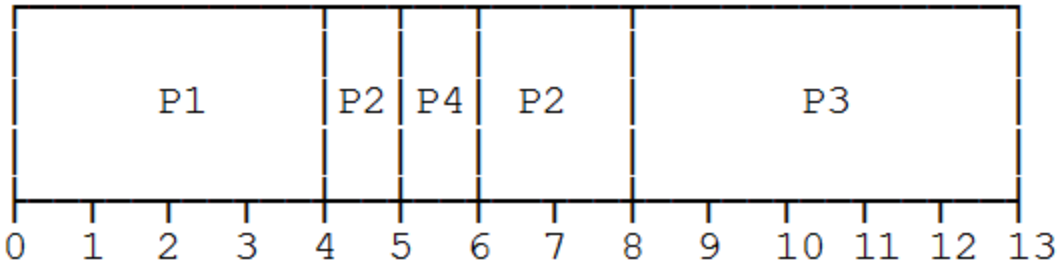


---

2. Next, you will implement and evaluate the **Shortest-Remaining-Time-First** (SRTF) Scheduling: Recall, this is the preemptive version of SJF scheduling that was discussed in class. If two processes have the same remaining time, use the process Id to break the tie (process with smaller Id goes first).

Use your implementation of the SRTF policy to compute the following for processes in "jobs.txt". (When executed, your program should print the following pieces of information on screen for the MAS algorithm.) (**40 points**)

a) Average **turnaround time** for all the jobs.
b) Average **waiting time** for all the jobs.
c) Besides, report the order in which processes were allowed to execute on the CPU along with the process' final termination time. For instance, for the following scheduling of processes, your program should print: 1(4), 2(8), 4(6), 2(8), 3(13).



---

3. Prepare a "README.txt" file for your submission. The file should contain the following: (**5 points**)

a) Instructions for compiling and executing each of your program(s) on the CS1 server. Include an example command line for the same.

b) If your implementation does not work, you should also document the problems in the README file, preferably with your explanation of why it does not work and how you would solve it if you had more time.

4. I recommend, you comment your code well. The best way to go about it is to write comments while coding.

Additional Assumptions and Requirements

- Your program must detect invalid command lines. If the command line is invalid, abort the program with an error message.
- If the input file does not exist (or does not open), abort the program with an error message.
- You may assume the input file is well formatted:
  - Each line consists of three integers followed by a newline (including the last line).
  - The process id is unique for each line (and is greater than zero).
  - The arrival time is greater than or equal to zero.
  - The burst time is strictly greater than zero.
  - The priority is greater than or equal to zero.
- The input file will contain at least one line. Beyond this, you cannot make any assumptions on how long the input file is.
- You may not assume anything about the order of line in the input file. The lines are not necessarily ordered by process id and/or their arrival times.
- If there is a tie, select the process with the lower process id.
- Assume that context switch overhead is zero.
- The average waiting time must be displayed as a floating point number.

Implementation Notes

- If you need help on using command line parameters in C++, please consult this link (http://www.cplusplus.com/articles/DEN36Up4/) for a brief tutorial.
- You may use STL for this assignment.

What should you submit?

Submit your assignment as a zip file on Canvas. The zip file should contain:

1. All your code files and any other files that might be needed for executing your code.
2. README.txt.