Machine Problem 5: Scheduler

DUE: Friday March 24, 2017, 11:59pm

I. Objective: In this machine problem, you are going to **simulate** several scheduling algorithms (or variations) discussed in class. You will write a C++ program to implement a simulator with different scheduling algorithms. The simulator selects a task (process) to run from the ready queue based on a scheduling algorithm. Since the project intends to **simulate** a CPU scheduler, it does not require any actual process creation or execution. When a task is scheduled for execution, the simulator will simply print information pertaining to the task at the specific instance of time.

II. Description: The selected scheduling algorithms in this machine problem are (a) First Come First Served (FCFS), Shortest Remaining Time First (SRTF), Round Robin (RR), and Multi-Level Feedback Queue (MLFQ). The scheduling algorithms are described below.

Let us take a closer look at the implementation specifications of each scheduling algorithm:

A). First Come First Served:

- The order of execution of the processes is exactly as the name FCFS implies.
- When multiple processes have the same arrival time, you can randomly choose the order of execution of the processes.
- The CPU starts execution when it sees the first process in the ready queue. That arrival time is the “system time” you are going to use to count the time elapsed. The system time starts with 0 which is the time the CPU starts working on the first process.
- Assume there is not any delay for the following cases(This assumption applies to the other algorithms as well):
    - Between CPU sees a process and it starts working on that process.
    - Process switch for a same CPU.

B). Round Robin:

Several aspects you should notice here:

- Round Robin scheduling policy is one where the CPU executes any given process for a time quantum specified for the algorithm. Once the process time quantum is up, the CPU moves to the next process. If the process completes before its time quantum expires, the CPU moves to the next process with a fresh time quantum allocation.
- The order in which the processes are handled follows the **FCFS** policy, which means you should sort the processes by their arrival time. Round-Robin policy will be applied to the sorted processes in that order.

- For the case where multiple processes arrive at the same time, you can randomly pick them in any order until they are all served.
- Similar to **FCFS**, CPU starts counting time elapsed when it sees the first process. And the initial time point is 0.

C). Shortest Remaining Time First (SRTF):

- Shortest Remaining Time First (SRTF) algorithm schedules a process for execution whose remaining execution (burst) time is shortest among all processes up for consideration in the ready queue.
- The order in which the processes are handled follows previous approaches, which means you should sort the processes by their arrival time. Once the order of the processes is arranged, SRTF policy will be applied to the sorted process in that order. For example, at a certain time *t* that the CPU just finished a job, at the same time, the CPU finds that multiple processes are waiting, then it will immediately starts working with the process which has the least amount of execution time. When it is working that process, there might be other processes coming on the way. It will do similar thing after it completes its current job.

- We are assuming a **preemptive SRTF policy**. To implement this, assume the smallest unit time in the system is 1 second, and it is the minimum time a CPU can work on a process. Every 1 second, the scheduler checks if there is any other process arrival. If so, the scheduler compares the remaining time for the current process against the burst time of the newly arrived process and of all other previously existing processes in the ready queue. It will choose the process with shortest remaining execution time for execution. It will keep working in this fashion. Here is the scheduling result for the processes in the previous file.

D). Multi-Level Feedback Queue Scheduling algorithm.

Our Multi-Level Feedback Queue (MLFQ) has the following attributes and follows simple rules as described below:

Attributes:

1. Total 4 levels of feedback queues
2. **Queues at the top-3 levels implement a Round Robin (RR) scheduling policy with time quantum of 4, 8, and 16 seconds respectively for Levels 1, 2 and 3. The "Queue" here does not always mean the data structure queue in this project, you will see that not all the policies use the data structure queue in the provided code.**
3. Queue at Level 4 implements First Come First Served (FCFS) scheduling policy.
4. The Levels 1-3 RR queues are 20 entries deep whereas the bottom Level 4 FCFS queue has infinite capacity.

5. For the first three levels, the scheduler moves from one level to the next by operating at specified time quantum granularity. Specifically, the scheduler starts at Level 1 and handles all its processes a quantum of 4. Once it is done with Level 1, it moves to handle processes at Level 2 where it will execute a round through all its processes for a quantum of 8, and then move to Level 3 to execute a quantum of 16 for all its processes.
6. Once done with Level 3, the scheduler attends to processes queued at Level 4 where it executes them in a non-preemptive FCFS fashion. **There is a limit of 5 processes executed at Level 4 in one pass.**
7. Once done with the pass through Level 4 (5 or fewer processes), the scheduler circulates back to Level 1 and repeats the protocol.
8. Eventually, all processes will go to the FCFS level. But the later arrival processes cannot break the basic FCFS rule to do their job first.

Rule 1: When a process enters the system, it is added to the tail of the Level 1 queue. If Level 1 queue is full, the process is added at the tail of the next Level queue. The process moves to Level 3 if Level 2 queue is full. If the queues at the top three levels are all full, the incoming process will be added at the tail of the lowest level Level 4 queue since it has infinite size. The order the process will be added also follows FCFS policy, which means the CPU always put the processes came earlier to available spots in the job queue.

Rule 2: Once a process uses up its time quantum at a given level, it moves down one level. If the queue at that level is full, it moves down one more level and onwards till it finds an empty spot.

III. Implementation:

You will be given the skeleton code for this project. All implementation must be done in C++.

Process Information: The task information will be read from an input file. The format is as follows for the columns with each row representing information about a process:

pid *arrival_time* *burst_time*

All of fields are integer type where
pid is a unique numeric process ID
arrival_time is the time in seconds when the task arrives in the ready queue
burst_time is the CPU time in seconds required by a task

All time units can be assumed to be in seconds. Each filed should be separated by a tab. Make sure you make this process information file in Unix/Linux setting as the format in Windows is different, which can simply mess your intention in reading the file. Don’t put the title for each filed there. A sample file will be provided to you.

Command-Line Usage and Examples

Usage: <program name> --input_file <file> --policy <policy> --quantum <time_quantum> will be used as command to run the program. input_file is the file name with task information described above. FCFS, RR, SRTF, and MFQ are names of scheduling algorithms. The time_quantum only applies to RR. **FCFS is nonpreemptive while RR and SRTF are preemptive**. See following table for more examples:

Examples	**Description**
<program name> --input_file task.txt –policy FCFS	**FCFS scheduling with the data file "input_file"**
<program name> --input_file task.txt –policy RR –quantum 5	**RR scheduling with the data file "input_file" and time quantum 5 seconds**
<program name> input_file task.txt –policy SRTF	**SRTF scheduling with the data file "input_file"**
<program name> input_file task.txt –policy MLFQ	**MLFQ scheduling with the data file "input_file"**

You can also add your own functions or variables to compute the waiting time or response time. But that's only limited to compute the waiting time and response time. You should think how to use the given infrastructure to do the job here by reading the comment for the functions in the code.

IV. Results & Report (Rubric specified in [] braces):

The following are the expectations for the machine problem deliverables:

1. **[120 points: 4 algorithms, 30 points each – 20 points for correct output and 10 points for comprehensive printed output messages]** For each algorithm, compute the average waiting time and average response time based on the given parameters in the source code. The average wait and response time are as defined in the class presentation (W6: Scheduling – Slide on Scheduling Metrics pp. 14). You will be graded on (a) the correctness of the average wait and response time, and (b) the explicit printed output as shown in the examples in Section II.

   TURN IN: A zipped file "MP5out.z" containing: (a) Your scheduler code in C++ (b) Makefile
   Make sure your program is generic by changing those Macro parameters in the MLFQ.h
   We will do different tests based on changing those values. You can assume the top 3 levels have the same size. If your program works only for the initial parameters, then you cannot do the testing based on changing quantum or queue size.

2. **[40 points: 20 points each for RR and MLFQ]** Repeat the exercise in #1 above for RR and MLFQ scheduling algorithms but with varying Round Robin Quantum value. For MLFQ, you may assume different quantum for Level1 and then scale Level 2 and 3 quanta in exact ratio as in the original spec. So for example, the original spec says Levels 1, 2, and 3 with time quantum of 4, 8, and 16 respectively. So if you choose 5 points of 4, 6, 8, 10, 12 for Level1, then the quanta for Levels 2 and 3 will respectively be Level2: 8, 12, 16, 20, 24 and Level3:

16, 24, 32, 40, 48. Chart out a response to the average waiting time and average response time (x-axis: Quantum and y-axis: average wait time and average response time.)

<u>TURN IN:</u> A PDF file "MP5chart.pdf" containing plotted charts for the exercise listed above. There will be 4 charts, each of RR and MLFQ scheduling algorithms will have their own average wait versus quantum and average response time versus quantum.

3. **[40 points]** Think about what other changes (for example, adding promotion policy to prevent starvation, dynamic policy to change time quantum to keep fairness and decrease context switch between processes) you can make based on the given program to make the mini CPU work with higher performance. Report your other findings as well in the report.

   <u>TURN IN</u>: A PDF file "MP5report.pdf" containing your analysis and commentary for the findings mentioned above in #3

   Also, you do need to demo for this project since we do not have a grading script for it. To get the points, please do the demo. We will use our own test files. You can test with whatever files you make. Use proper data there.

Sample Outputs:

Run FCFS on the given sample process information file Task.txt, you will have something like: (You can use I/O redirection to redirect your output to a file like this: ./schedule --input_file Task.txt --policy SRTF > srtf_task.txt for your later calculation on average waiting and response time)

```
System Time[0].........Process[PID=1] is Running
System Time[1].........Process[PID=1] is Running
System Time[2].........Process[PID=1] is Running
System Time[3].........Process[PID=1] finished its job!
System Time[3].........Process[PID=2] is Running
System Time[4].........Process[PID=2] is Running
System Time[5].........Process[PID=2] is Running
System Time[6].........Process[PID=2] is Running
System Time[7].........Process[PID=2] finished its job!
System Time[7].........Process[PID=3] is Running
System Time[8].........Process[PID=3] is Running
System Time[9].........Process[PID=3] is Running
System Time[10].........Process[PID=3] is Running
System Time[11].........Process[PID=3] is Running
System Time[12].........Process[PID=3] finished its job!
System Time[12].........Process[PID=4] is Running
System Time[13].........Process[PID=4] is Running
System Time[14].........Process[PID=4] is Running
System Time[15].........Process[PID=4] is Running
System Time[16].........Process[PID=4] is Running
System Time[17].........Process[PID=4] is Running
System Time[18].........Process[PID=4] finished its job!
System Time[18].........Process[PID=5] is Running
System Time[19].........Process[PID=5] is Running
System Time[20].........Process[PID=5] is Running
System Time[21].........Process[PID=5] is Running
System Time[22].........Process[PID=5] is Running
System Time[23].........Process[PID=5] finished its job!
System Time[23].........Process[PID=6] is Running
System Time[24].........Process[PID=6] is Running
System Time[25].........Process[PID=6] is Running
System Time[26].........Process[PID=6] is Running
System Time[27].........Process[PID=6] is Running
System Time[28].........Process[PID=6] is Running
System Time[29].........Process[PID=6] is Running
System Time[30].........Process[PID=6] is Running
System Time[31].........Process[PID=6] is Running
System Time[32].........Process[PID=6] finished its job!
System Time[32].........Process[PID=10] is Running
System Time[33].........Process[PID=10] is Running
System Time[34].........Process[PID=10] is Running
System Time[35].........Process[PID=10] is Running
System Time[36].........Process[PID=10] is Running
System Time[37].........Process[PID=10] is Running
System Time[38].........Process[PID=10] is Running
System Time[39].........Process[PID=10] is Running
System Time[40].........Process[PID=10] finished its job!
System Time[40].........Process[PID=7] is Running
System Time[41].........Process[PID=7] is Running
System Time[42].........Process[PID=7] is Running
System Time[43].........Process[PID=7] is Running
System Time[44].........Process[PID=7] finished its job!
System Time[44].........Process[PID=9] is Running
System Time[45].........Process[PID=9] is Running
System Time[46].........Process[PID=9] is Running
System Time[47].........Process[PID=9] is Running
System Time[48].........Process[PID=9] is Running
System Time[49].........Process[PID=9] finished its job!
System Time[49].........Process[PID=8] is Running
System Time[50].........Process[PID=8] is Running
System Time[51].........Process[PID=8] finished its job!
```

Run RR on the given sample process information file Task.txt, you will have something like:

```
System Time[0].........Process[PID=1] is Running
System Time[1].........Process[PID=1] is Running
System Time[2].........Process[PID=1] is Running
System Time[3].........Process[PID=1] finishes its quantum
System Time[3].........Process[PID=1] finished its job!
System Time[3].........Process[PID=2] is Running
System Time[4].........Process[PID=2] is Running
System Time[5].........Process[PID=2] is Running
System Time[6].........Process[PID=2] finishes its quantum
System Time[6].........Process[PID=3] is Running
System Time[7].........Process[PID=3] is Running
System Time[8].........Process[PID=3] is Running
System Time[9].........Process[PID=3] finishes its quantum
System Time[9].........Process[PID=4] is Running
System Time[10].........Process[PID=4] is Running
System Time[11].........Process[PID=4] is Running
System Time[12].........Process[PID=4] finishes its quantum
System Time[12].........Process[PID=5] is Running
System Time[13].........Process[PID=5] is Running
System Time[14].........Process[PID=5] is Running
System Time[15].........Process[PID=5] finishes its quantum
System Time[15].........Process[PID=6] is Running
System Time[16].........Process[PID=6] is Running
System Time[17].........Process[PID=6] is Running
System Time[18].........Process[PID=6] finishes its quantum
System Time[18].........Process[PID=7] is Running
System Time[19].........Process[PID=7] is Running
System Time[20].........Process[PID=7] is Running
System Time[21].........Process[PID=7] finishes its quantum
System Time[21].........Process[PID=10] is Running
System Time[22].........Process[PID=10] is Running
System Time[23].........Process[PID=10] is Running
System Time[24].........Process[PID=10] finishes its quantum
System Time[24].........Process[PID=9] is Running
System Time[25].........Process[PID=9] is Running
System Time[26].........Process[PID=9] is Running
System Time[27].........Process[PID=9] finishes its quantum
System Time[27].........Process[PID=8] is Running
System Time[28].........Process[PID=8] is Running
System Time[29].........Process[PID=8] finishes its quantum
System Time[29].........Process[PID=8] finished its job!
System Time[29].........Process[PID=2] is Running
System Time[30].........Process[PID=2] finishes its quantum
System Time[30].........Process[PID=2] finished its job!
System Time[30].........Process[PID=4] is Running
System Time[31].........Process[PID=4] is Running
System Time[32].........Process[PID=4] is Running
System Time[33].........Process[PID=4] finishes its quantum
System Time[33].........Process[PID=4] finished its job!
System Time[33].........Process[PID=6] is Running
System Time[34].........Process[PID=6] is Running
System Time[35].........Process[PID=6] is Running
System Time[36].........Process[PID=6] finishes its quantum
System Time[36].........Process[PID=7] is Running
System Time[37].........Process[PID=7] finishes its quantum
System Time[37].........Process[PID=7] finished its job!
System Time[37].........Process[PID=9] is Running
System Time[38].........Process[PID=9] is Running
System Time[39].........Process[PID=9] finishes its quantum
System Time[39].........Process[PID=9] finished its job!
System Time[39].........Process[PID=5] is Running
System Time[40].........Process[PID=5] is Running
System Time[41].........Process[PID=5] finishes its quantum
System Time[41].........Process[PID=5] finished its job!
System Time[41].........Process[PID=3] is Running
System Time[42].........Process[PID=3] is Running
System Time[43].........Process[PID=3] finishes its quantum
System Time[43].........Process[PID=3] finished its job!
System Time[43].........Process[PID=10] is Running
System Time[44].........Process[PID=10] is Running
System Time[45].........Process[PID=10] is Running
System Time[46].........Process[PID=10] finishes its quantum
System Time[46].........Process[PID=6] is Running
System Time[47].........Process[PID=6] is Running
System Time[48].........Process[PID=6] is Running
System Time[49].........Process[PID=6] finishes its quantum
System Time[49].........Process[PID=6] finished its job!
System Time[49].........Process[PID=10] is Running
System Time[50].........Process[PID=10] is Running
System Time[51].........Process[PID=10] finishes its quantum
System Time[51].........Process[PID=10] finished its job!
```

Run SRTF on the given sample process information file Task.txt, you will have something like:

```
System Time[0].........Process[PID=1] is Running
System Time[1].........Process[PID=1] is Running
System Time[2].........Process[PID=1] is Running
System Time[3].........Process[PID=1] finished its job!
System Time[3].........Process[PID=2] is Running
System Time[4].........Process[PID=2] is Running
System Time[5].........Process[PID=2] is Running
System Time[6].........Process[PID=2] is Running
System Time[7].........Process[PID=2] finished its job!
System Time[7].........Process[PID=7] is Running
System Time[8].........Process[PID=7] is Running
System Time[9].........Process[PID=7] is Running
System Time[10].........Process[PID=7] is Running
System Time[11].........Process[PID=7] finished its job!
System Time[11].........Process[PID=8] is Running
System Time[12].........Process[PID=8] is Running
System Time[13].........Process[PID=8] finished its job!
System Time[13].........Process[PID=3] is Running
System Time[14].........Process[PID=3] is Running
System Time[15].........Process[PID=3] is Running
System Time[16].........Process[PID=3] is Running
System Time[17].........Process[PID=3] is Running
System Time[18].........Process[PID=3] finished its job!
System Time[18].........Process[PID=9] is Running
System Time[19].........Process[PID=9] is Running
System Time[20].........Process[PID=9] is Running
System Time[21].........Process[PID=9] is Running
System Time[22].........Process[PID=9] is Running
System Time[23].........Process[PID=9] finished its job!
System Time[23].........Process[PID=5] is Running
System Time[24].........Process[PID=5] is Running
System Time[25].........Process[PID=5] is Running
System Time[26].........Process[PID=5] is Running
System Time[27].........Process[PID=5] is Running
System Time[28].........Process[PID=5] finished its job!
System Time[28].........Process[PID=4] is Running
System Time[29].........Process[PID=4] is Running
System Time[30].........Process[PID=4] is Running
System Time[31].........Process[PID=4] is Running
System Time[32].........Process[PID=4] is Running
System Time[33].........Process[PID=4] is Running
System Time[34].........Process[PID=4] finished its job!
System Time[34].........Process[PID=10] is Running
System Time[35].........Process[PID=10] is Running
System Time[36].........Process[PID=10] is Running
System Time[37].........Process[PID=10] is Running
System Time[38].........Process[PID=10] is Running
System Time[39].........Process[PID=10] is Running
System Time[40].........Process[PID=10] is Running
System Time[41].........Process[PID=10] is Running
System Time[42].........Process[PID=10] finished its job!
System Time[42].........Process[PID=6] is Running
System Time[43].........Process[PID=6] is Running
System Time[44].........Process[PID=6] is Running
System Time[45].........Process[PID=6] is Running
System Time[46].........Process[PID=6] is Running
System Time[47].........Process[PID=6] is Running
System Time[48].........Process[PID=6] is Running
System Time[49].........Process[PID=6] is Running
System Time[50].........Process[PID=6] is Running
System Time[51].........Process[PID=6] finished its job!
```

Run MLFQ on the given sample process information file Task.txt, you will have something like (Since the given file is relatively small, I make this example based on these parameters: assuming level 1, 2, 3 size is 2 and their quantum is 2, 3, 4, respectively; and the last level each time execute 2 jobs):

```
System Time[0].........First Level Queue Starts Work
System Time[0].........Process[PID=1] Starts working
System Time[0].........Process[PID=1] is Running
System Time[1].........Process[PID=1] is Running
System Time[2].........Process[PID=1] finished its Quantum
degrading [PID=1]
System Time[2].........Process[PID=2] Starts working
System Time[2].........Process[PID=2] is Running
System Time[3].........Process[PID=2] is Running
System Time[4].........Process[PID=2] finished its Quantum
degrading [PID=2]
System Time[4].........Second Level Queue Starts Work
System Time[4].........Process[PID=3] Starts working
System Time[4].........Process[PID=3] is Running
System Time[5].........Process[PID=3] is Running
System Time[6].........Process[PID=3] is Running
System Time[7].........Process[PID=3] finished its Quantum
degrading [PID=3]
System Time[7].........Process[PID=4] Starts working
System Time[7].........Process[PID=4] is Running
System Time[8].........Process[PID=4] is Running
System Time[9].........Process[PID=4] is Running
System Time[10].........Process[PID=4] finished its Quantum
degrading [PID=4]
System Time[10].........Third Level Queue Starts Work
System Time[10].........Process[PID=5] Starts working
System Time[10].........Process[PID=5] is Running
System Time[11].........Process[PID=5] is Running
System Time[12].........Process[PID=5] is Running
System Time[13].........Process[PID=5] is Running
System Time[14].........Process[PID=5] finished its Quantum
degrading [PID=5]
System Time[14].........Process[PID=6] Starts working
System Time[14].........Process[PID=6] is Running
System Time[15].........Process[PID=6] is Running
System Time[16].........Process[PID=6] is Running
System Time[17].........Process[PID=6] is Running
System Time[18].........Process[PID=6] finished its Quantum
degrading [PID=6]
System Time[18].........FCFS Queue Starts Work
System Time[18].........Process[PID=7] is Running
System Time[19].........Process[PID=7] is Running
System Time[20].........Process[PID=7] is Running
System Time[21].........Process[PID=7] is Running
System Time[22].........Process[PID=7] finished its job!
System Time[22].........Process[PID=10] is Running
System Time[23].........Process[PID=10] is Running
System Time[24].........Process[PID=10] is Running
System Time[25].........Process[PID=10] is Running
System Time[26].........Process[PID=10] is Running
System Time[27].........Process[PID=10] is Running
System Time[28].........Process[PID=10] is Running
System Time[29].........Process[PID=10] is Running
System Time[30].........Process[PID=10] finished its job!
System Time[30].........FCFS Queue Starts Work
System Time[30].........Process[PID=9] is Running
System Time[31].........Process[PID=9] is Running
System Time[32].........Process[PID=9] is Running
System Time[33].........Process[PID=9] is Running
System Time[34].........Process[PID=9] is Running
System Time[35].........Process[PID=9] finished its job!
System Time[35].........Process[PID=8] is Running
System Time[36].........Process[PID=8] is Running
System Time[37].........Process[PID=8] finished its job!
System Time[37].........FCFS Queue Starts Work
System Time[37].........Process[PID=1] is Running
System Time[38].........Process[PID=1] finished its job!
System Time[38].........Process[PID=2] is Running
System Time[39].........Process[PID=2] is Running
System Time[40].........Process[PID=2] finished its job!
System Time[40].........FCFS Queue Starts Work
System Time[40].........Process[PID=3] is Running
System Time[41].........Process[PID=3] is Running
System Time[42].........Process[PID=3] finished its job!
System Time[42].........Process[PID=4] is Running
System Time[43].........Process[PID=4] is Running
System Time[44].........Process[PID=4] is Running
System Time[45].........Process[PID=4] finished its job!
System Time[45].........FCFS Queue Starts Work
System Time[45].........Process[PID=5] is Running
System Time[46].........Process[PID=5] finished its job!
System Time[46].........Process[PID=6] is Running
System Time[47].........Process[PID=6] is Running
System Time[48].........Process[PID=6] is Running
System Time[49].........Process[PID=6] is Running
System Time[50].........Process[PID=6] is Running
System Time[51].........Process[PID=6] finished its job!
Scheduling for all tasks has been done.
```

Finally, in your report, you should put the computed average waiting time and other time metric in the table like this(this is just an example, not based on the results above): This is for a process information file. Plot based on the results from RR and MLFQ

FCFS	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7
Average Waiting time							
Average Response Time							

RR, Quantum=2, 3, 4	Process 1			Process 2			Process 3			Process 4			Process 5			Process 6			Process 7		
	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Average Waiting time																					
Average Response Time																					

MLFQ, Quantum top three levels: 2, 3, 4	Process 1			Process 2			Process 3			Process 4			Process 5			Process 6			Process 7		
	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Average Waiting time																					
Average Response Time																					