CSCI 447 - Operating Systems, Spring 2024
Homework 3 : CPU Scheduling
Due Date: See Canvas

~~Collaboration~~ Policy

Your homework submissions must be yours. You may chat with other students on a high-level about topics and concepts, but you cannot share, disseminate, co-author, or even view, other students' code. If any of this is unclear, please ask for further clarification.

1 Free Response Questions

Upload a pdf document to Canvas for the *Homework 3 - questions* submission. Please be concise, but thorough. Single line answers do not suffice, but 1-page long narrative answers are too long. If a calculation is involved, please show your work. Correct answers without proof of work will receive partial credit only.

Q1 : 5 points

Explain why round robin scheduling with a very small (< 2ms) time quantum can result in a slow overall execution time of multiple processes versus when the time quantum is larger (approximately 20ms). **Explicitly provide multiple reasons in support of your stance.**

Q2 : 5 points

Assume the following processes, their arrival times, and needed burst times. Further assume nonpreemptive scheduling. The OS makes dispatch decisions at the moment when the CPU is free, taking into account all information that it has at that time.

- P_1, arrival: 0.0, burst: 8
- P_2, arrival: 0.4, burst: 4
- P_3, arrival: 1.0, burst: 1

What is the average turnaround time for the three processes, if FCFS is used? If SJF is used?

Compute the average turnaround time if the CPU is left idle for the first 1 unit of time, and thereafter SJF is used. In other words, P_1 and P_2 wait from t=0 to t=1.

Q3 : 5 points

Which of the following scheduling algorithms could result in starvation? For each algorithm that you identify, provide the priorities and/or requests for two or more processes, which would result in one (or more) of them being starved.

a. First-come, first-served
b. Round robin
c. Shortest job first
d. Priority

Q4 : 5 points

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

For the above segment table, what are the physical addresses for the following logical addresses?

a. 0,430

b. 3,400

c. 1,10

d. 4,112

e. 2,500

2 Coding Tasks : 2 points for setup, 54 points for the program

For the coding task, you will calculate the run-time of a series of processes under different feedback queue conditions, for a single CPU system. Your task is to implement a multilevel feedback queue CPU scheduling algorithm, to be used with an input file that provides simulated arrival times and instruction histories for one or more processes. Your goal is to determine:

- The best operational parameters of your feedback queue
- Whether a preemptive or nonpreemptive scheduler provides better performance.

2.1 Setup

Create and checkout a new branch on your gitlab for this assignment. Name the branch *homework3*. **When creating a new branch, ALWAYS do that from the master branch.** In the *homework3* branch, create a new directory, *homework3*. All work for this homework assignment should be done in the *homework3* branch and *homework3* directory.

2.2 Multilevel Queue Structure

The multilevel feedback queue algorithm must use two queues, here referred to as Queue A and Queue B. The algorithm should work similar (but not exactly like ... see bolded text in this paragraph) to a round robin strategy, executing first all the processes in queue A. Only when queue A is empty, the scheduler will dispatch processes from queue B. In this way, queue A acts as a high priority queue, and queue B as a low priority queue. When a process is first added to the ready queue, it is enqueued into queue B. **Due to the behavior of promotion (see below), when a process is dispatched from queue B, that process might eventually be elevated to queue A, in which case the scheduler should next begin dispatching processes from queue A**.

The two queues must implement the following scheduling algorithms:

- Queue A implements FCFS
- Queue B implements priority scheduling

2.3 Dispatch, and Promotion

When a process uses all of its allotted quantum for that turn to the CPU, it is returned to the tail-end of the queue from which it was dispatched; the only exception is if the process is promoted. A process is promoted from B to A if it uses the CPU for less than its assigned quantum in each of 3 successive CPU bursts, which includes occasions when the process is preempted, or when the process requests an IO operation. Promotion and context switching are both instantaneous, and assume that they do not incur a time cost. Once a process is promoted to Queue A, it remains in that queue.

2.4 "Simulation" Input File

Your program must receive as input, provided as a command line argument, a plain text file, with simulated instructions that must be scheduled and dispatched. The input plain text file that "is" the simulation has the following format:

```
P1002:2
arrival_t:34
exe:5
io:3
exe:4
io:5
exe:45
terminate
```

where the first line specifies a Process ID (1002 in this case), and the "2" to the right of the colon specifies a priority, where higher numbers specify a HIGHER priority. The `arrival_t` line specifies the dimensionless time at which the process arrives to the ready queue. You can assume that process information is provided in the input file ordered by correct arrival time, but process IDs are not necessarily sequential. Everything after the `arrival_t` line represents the instructions for that process. A sample file is available in Canvas.

- The words `exe` and `io` specify execution (needing CPU) and IO operations (not needing CPU), followed by numbers that designate the dimensionless units of time needed by those instructions.
- It takes 1 unit of time for the OS to "decode" an `io` and `exe` instruction. For example, `io:5` would incur 1 unit of time consumed by the CPU for the OS to decode that instruction, and then 5 units of time (not consuming CPU time) performing the IO operation. And `exe:3` would incur 1 unit of time to decode the instructions, and 3 units of time to execution the operation (assuming the process is not preempted or the quantum is high enough).
- The time that it takes for an `exe` or `io` instruction to be decoded consumes CPU time, and that consumption is deducted from the quantum.
- The system does NOT have an IO queue(s). Multiple IO operations (from different Processes) can be running in parallel. However, if there are back-to-back IO operations in a single process, then the second IO operation is not decoded until the first IO operation completes. Recall that instructions in a single process are executed in series.

- It takes 1 unit of time TOTAL for the OS to "decode" and "execute" a `terminate` instruction.
- It takes 1 unit of time for the OS to "decode" an `exe` instruction each time that instruction must be performed, regardless of how many time units that `exe` instruction requires. If an `exe` instruction is preempted, and/or evicted because it has used the entirety of its quantum, the next time that the `exe` instruction is executed, it must be decoded again.
- If an `exe` instruction is preempted, that `exe` instruction's required time is decremented by the amount of time that the `exe` instructions was "in" the CPU.
- A process contains at least one `exe` or `io` instruction, but there is no upper limit.
- The word `terminate` specifies the end of the instructions for a process.

2.5 Implementation Details

Your Simulation program should have the capability to run in either preemptive or non-preemptive mode (implement the non-preemptive mode first). If preemption is turned on, it applies ONLY to queue B. If a process arrives (enters Queue B), and the priority of the process is higher than the priority of the currently executing process, the process should preempt the currently executing process. The only exception to this is if the process that is in the CPU was dispatched from Queue A, in which case the process in the CPU should NOT be preempted. If a process is preempted, it is returned to the appropriate Queue A or Queue B, unless it is promoted, in which case it is moved to the tail end of Queue A.

Preemption, promotion, prioritizing, etc., are all done at the START of a time tick. This means, that if a process **P45** arrives at $t = 72$, and the currently executing process **P20** has a lower priority than **P45**, and preemption is enabled, then process **P20** is preempted, and does NOT consume any more CPU time. Instead, **P45** runs from $t = 72$ until $t = 73$ (and perhaps longer).

One of your tasks is to explore the best values of quanta for queues A and B. The length of these two quanta are the parameter values for the scheduling algorithm, and your goal is to specify those quanta that result in an ideal system where wait times are minimized.

2.6 Invocation

Please use the skeleton of the `Simulation.c` file to implement your solution. The *Simulation.c* program should be modified so that the *Simulation* executable accepts 4 command-line arguments, in the following order :

1. Name of input file for parsing of simulated processes
2. quantum value (int) for queue A
3. quantum value (int) for queue B
4. preemption (1=yes, 0=no)

For example: `./Simulation aFile.txt 4 32 0`

Complete the main function in *Simulation.c*, and implement the 2-level feedback ready queue. You may write additional classes, use custom data structures, functions, etc. to tally and keep track of processes as they are executing, but it is the *Simulation.c* program that will be invoked

and run to test your implementation.

Because it takes at least 1 unit of time to decode `io`, `exe`, and `terminate` instructions, your program will be tested where the quanta are 2 or more. Testing will be done for quanta values up to and including 10.

2.7 Detailed Example

Assume the following input file *sampleFile.txt*:

```
P1700:10
arrival_t:1
io:2
exe:3
io:5
exe:4
terminate
P456:5
arrival_t:3
exe:3
terminate
```

Further assume that the quantum for Queue A is 7, the quantum for Queue B is 3, and no preemption. In other words, invocation would be: `./Simulation sampleFile.txt 7 3 0`

Figure 1 shows how the processes in *sampleFile.txt* would be simulated.

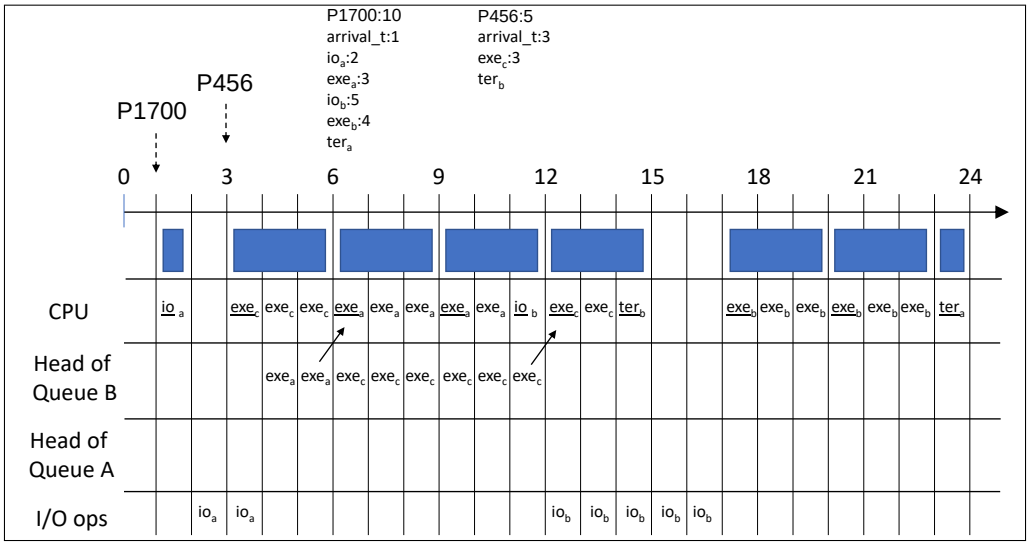

Figure 1: Simulation of *sampleFile.txt*. Underlined specifies decoding, and the IO ops row designates an active IO operation. Subscripts ($_{a,b}$, *etc.*) have been added to distinguish between different items of the same instruction type. Blue boxes specify quanta used. Only the heads of queues are shown. Dotted arrows designate arrival times. Diagonal arrows designate dispatch. Instruction exe_a is evicted at t=9, is enqueued to Queue B, but because of priority, goes to the front of Queue B, is dispatched again, hence the decoding that starts at t=9.

2.8 Program Output and Analysis

To help you determine the best quanta values, and to aid in your analysis of your queue, your program must print to the screen the following (in this order):

- **Start time and end time**: the start time of the simulation, and the time at which the last process terminates
- **Num Processes completed**: the count of processes completed
- **Num instructions completed**: the count of instructions completed
- **Ready time**: the average, maximum, and minimum times that a process(es) spend in the ready (multilevel feedback) queue.
- **Completion Order**: the IDs of the processes in the order that they finish, their finish times, the total time that each process spent waiting in the multilevel queue, and which queue they were dispatched from when they completed.

A sample output (numbers are made-up, but yours shouldn't be!) is the following:

```
Start/end time:  0, 467
Processes Completed:  4
Instructions Completed:  23
Ready time average:  0.41
Ready time maximum:  2
Ready time minimum:  1
P1001 time_completion:16 time_waiting:4 termination_queue:A
P1007 time_completion:22 time_waiting:7 termination_queue:B
P1011 time_completion:30 time_waiting:11 termination_queue:B
P1002 time_completion:36 time_waiting:8 termination_queue:A
```

2.9 Analysis

Compose a plain text file, *SimulationParams.txt*, in which you analyze and justify the best use of quantum values so that throughput is maximized and ready times (and indirectly delay and starvation) are minimized. Provide tallies of your program for different simulations. **It is not enough for you to run your simulation a few times on only one or two input files, and to make your assessment based on that limited data. Be systematic. If you are unable to correctly code the entirety of the feedback queue, report on and analyze those parts of the queue that you were able to code.**

3 Submission and Rubric

Please push to the homework3 branch, into the homework3 directory, the following :

- A revised *Simulation.c*
- Any other custom .c and/or .h file(s) – if you needed to create and use a custom data structure(s)
- A Makefile (see lab3), which if invoked using `make` will generate a *Simulation* executable
- A short report, 1 "page" maximum, a plain text file, called *SimulationParams.txt*, that includes your findings on the (optimal) parameter values for the scheduling algorithm under preemptive and non-preemptive modes. **Include tallies of run-times and stats, and justify your reasoning for your choice of optimal quanta values, and preemption or no preemption**.

Written questions	20
Programming Task : setup (git), including branch	1
Programming Task : .o nor executable file(s) have NOT been pushed to your git branch	1
Programming Task : main routine correctly accepts command-line arguments	5
Programming Task : scheduler algorithm implemented	5
Programming Task : entirety of code compiles	5
Programming Task : a Makefile that compiles the program when `make` is invoked	5
Programming Task : correctness of scheduler algorithm; at least a half dozen unit cases will be used to test your queue, starting with input files with a single process where preemption won't happen, and proceeding to an input file with thousands of processes, with preemption enabled	24
Programming Task : quality of your code, including choice of data structures, efficiency, readability, commenting, etc.	5
Programming Task : completeness of your *SimulationParams.txt* data file, including a (brief) discussion, explaining your selection of the optimal parameters	5
Total	76 points