COMP3520 Operating Systems Internals Assignment 2: Multi-Level Queue Dispatcher

General Instructions

Assignment 2 is about a multi-level queue dispatcher and consists of two compulsory tasks:

1. **Revise the codes from Programming Exercises 7 or 8** to implement a Multi-Level Queue Dispatcher for a uniprocessor system (**65%**); and
2. Answer the discussion document questions (**35%**).

This assignment is an individual assignment. Whilst you are permitted to discuss this assignment with other students, the work that you submit must be your own work. You should not incorporate the work of other students (past or present) into your source code or discussion document.

You will be required to submit your discussion document to Turnitin for similarity checking as part of assignment submission. The examiner may use other similarity checking tools in addition to Turnitin. Your source code may also be checked.

Submit your source code (including **source code** and **makefile** together in a **zip/tar file)** and **discussion document** to the appropriate submission inboxes in the COMP3520 Canvas website.

Multi-Level Queue Dispatcher

In Assignment 2, you need to use a list of randomly generated jobs to schedule for execution. In addition to time of arrival and allowed execution time, each job also has an initial priority of 0, 1 or 2. A job's initial priority determines which ready queue it will be inserted into when it first arrives; there are three ready queues, one for each priority: Level-0, Level-1, and Level-2. The job dispatch list format is as follows:

<arrival time>, <cputime>,<priority>

0, 3, 0

2, 8, 2

4, 4, 0

6, 5, 0 ,

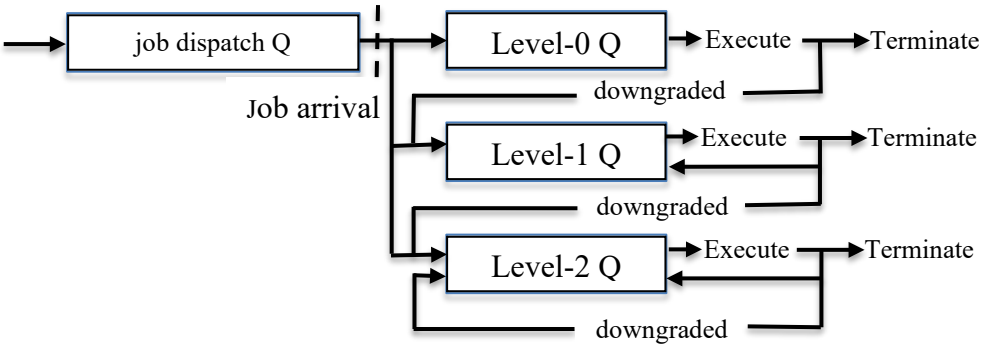
8, 5, 1

…

For this assignment, you need to use **ONE** Job Dispatch queue **and** a **THREE-LEVEL** ready queue to manage and dispatch jobs.

1. At the beginning, all jobs in the job dispatch list file are loaded into the job dispatch queue.

2. When a job has "arrived", it will be moved from the job dispatch queue to the appropriate ready queue according to the job's initial priority value. For example, a job with a priority of 0 will be moved to the Level-0 queue; a job with a priority of 1 will be moved to the Level-1 queue and a job with a priority of 2 will be moved to the Level-2 queue.



3. Jobs in the Level-0 queue are dispatched in a FCFS (First Come First Served) manner. When a job is dispatched, it is given a time-quantum of **t0** units of time (which can be considered as seconds in this assignment) for execution without interruption. If the job cannot complete execution within the time-quantum, it will be pre-empted and moved to the end (or tail) of the Level-1 queue (i.e., its priority is downgraded to 1).
4. Jobs in the Level-1 queue can be scheduled to run only when the Level-0 queue is empty. Jobs in the Level-1 queue are also dispatched in a FCFS manner. Each job is given a time-quantum of **t1** for execution at this level. When a new Level-0 job arrives, the currently running Level-1 job will be pre-empted immediately (regardless of whether it has used its allocated time-quantum as a Level-1 job or not). If the job has not used its time-quantum, it will be placed at the front (or head) of the Level-1 queue. It can then be scheduled to run again when the Level-0 queue becomes empty. If it cannot complete within the allocated time-quantum (i.e., if its accumulated run time at this level is equal to **t1**), the job will be moved to the end (or tail) of the Level-2 queue (i.e., its priority is downgraded to 2).
5. Jobs in the Level-2 queue can be scheduled to run only when both Level-0 and Level-1 queues are empty. When a new Level-0 or Level-1 job arrives, the currently running Level-2 job will be pre-empted immediately and placed at the front (or head) of the Level-2 queue. A job in the Level-2 queue is dispatched in a RR (Round Robin) manner; each Level-2 job is given a time-quantum of **t2** for execution. If it cannot complete within the time-quantum (i.e., if its accumulated run time at this level is equal to **t2**), the job will be pre-empted and placed at the end (or tail) of the same queue.
6. The dispatcher will immediately schedule another job for execution when the currently running process is terminated (completed) or suspended (pre-empted), and it always selects a job with the highest priority for execution. This may potentially cause starvation problems. To prevent starvation, the system sets a time **W**.
    a. When the job in front of the Level-1 queue has not been scheduled to run for **W** units of time since the last time it is pre-empted, all jobs in Level-1 and Level-2 queues are moved to the end (or tail) of the Level-0 queue (i.e., their priorities are upgraded to 0). Level-1 jobs are placed before Level-2 jobs and jobs at the same level will maintain their order.

   b. When the job in front of the Level-2 queue has not been scheduled to run for **W** units of time since the last time it is pre-empted, all jobs in the Level-2 queue are moved to the end (or tail) of the Level-0 queue with their original order intact.
7. Your program needs to correctly calculate and print out on the screen the average turnaround time, average waiting time and average response time.

Requirements

Source Code

1. Your program must ask the user to enter three integer values **t0**, **t1, t2** and **W** respectively.
2. You need to make necessary changes to those programs in Programming Exercises 7 or 8 to implement the Multi-Level Queue dispatcher.
3. Your program needs to correctly calculate and print out on the screen the average turnaround time, average waiting time and average response time.
4. Your program must be implemented in the C/C++ programming language.
5. Your source code needs to be properly commented and appropriately structured to allow another programmer who has a working knowledge of C/C++ to understand and easily maintain your code.
6. You need to include a *makefile* that allows for the compilation of your source code using the *make* command on the School of Computer Science computer servers. It is crucial that you ensure that your source code compiles correctly on the School of Computer Science servers. If your source code cannot be compiled on the School servers, marks will be heavily deducted.

Discussion Document

You are required to answer all assigned questions in a separate written document. The questions and requirements specific to the discussion document will be provided in a separate document.

Use of Writing Assistance and Generative AI Tools Policy

You are permitted to use automated writing and generative artificial intelligence (AI) tools to help you develop your program and write your report. You are required to complete and submit the "Automated Writing and Generative AI Tools Usage Form", even if you do not use writing assistance or generative AI tools at all. You will not lose marks for using these tools as long as you properly document your use of them.

Submission Requirements

1. Your submission must be made by 11:59pm on Friday, 1 November 2024 (Sydney time).
2. Create a **tar** or **zip** file that contains your **makefile** and **source files** (e.g., .c and .h files). DO NOT INCLUDE ANY OBJECT OR BINARY FILES.
3. Submit only one .tar or .zip file to the "Assignment 2 – Part 1" inbox.
4. Save your Discussion Document in pdf format.

5. Save your "Automated Writing and Generative AI Tools Usage Form" in pdf format.
6. Submit your Discussion Document and "Automated Writing and Generative AI Tools Usage Form" to the "Assignment 2 – Part 2" inbox as **separate** files.

Failure to follow these submission requirements may lead to loss of marks.

Other Matters

To maximize your chances of realizing your full potential in COMP3520, **please start work on this assignment promptly.**