

# CSC 369 Worksheet 5 Solution

August 18, 2020

1. I need to run randomly-generated problems with two jobs and two queues using file `mlfq.py` with I/O turned off, and compute the MLFQ execution trace for each.

Using the command `./mlfq.py -s 1 -m 10 -n 2 -j 2 -M 0`, we have

```
Job List:
Job  0: startTime  0 - runTime  2 - ioFreq  0
Job  1: startTime  0 - runTime  7 - ioFreq  0
```

with

- allotments for queue 1 is 1
- quantum length for queue 1 is 10
- allotments for queue 0 is 1
- quantum length for queue 0 is 10
- no priority boost

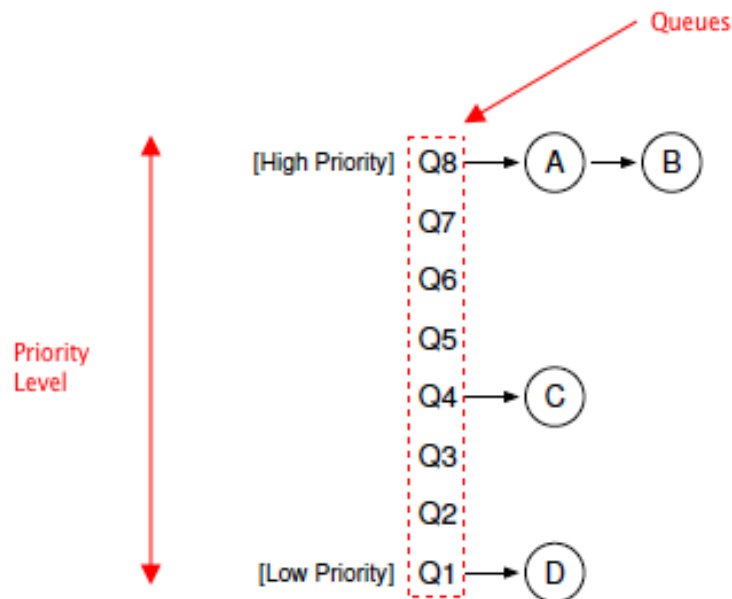
,

the execution trace is:

## Notes

- The reason why round robin doesn't occur despite  $\text{Priority}(A) = \text{Priority}(B)$  is because allotment of queue is 1 (i.e. only one job can be in a queue)
- **allotment** means the amount of something allocated to a person/object (i.e. the size of queue)
- `-m 10` sets the maximum runtime of a job to 10
- `-M 0` turns off I/O in `mlfq.py`

- -n 2 sets number of queues to 2
- -j 2 sets number of jobs to 2
- **Multi-level Feedback Queue (MLFQ):**
  - Is one of the most well-known approaches to scheduling
  - Does two things:
    - a) Optimizes turnaround time
    - b) Minimizes response time
  - Uses **priority level** and **Queues** to achieve it's goal
- **MLFQ Basic Rules:**
  - Jobs on same queue → Same priority
  - **Rule 1:** If  $\text{Priority}(A) > \text{Priority}(B)$ , A runs (B doesn't)
  - **Rule 2:** If  $\text{Priority}(A) = \text{Priority}(B)$ , A & B run in RR

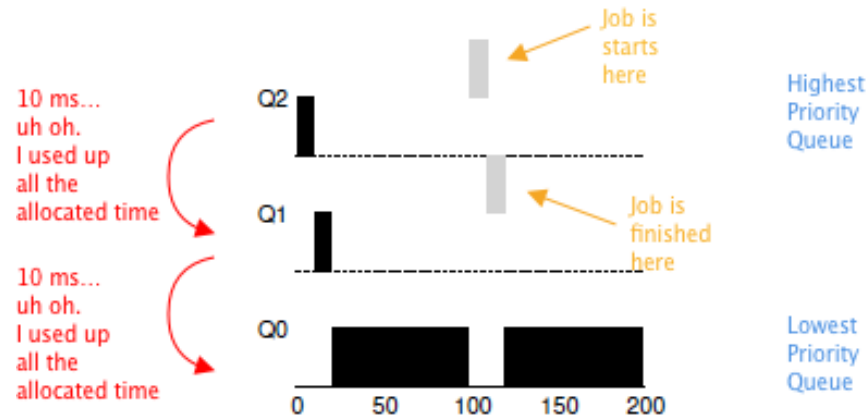


- **Attempt #1: How to Change Priority**
  - **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue)
  - **Rule 4a:** If a job uses up an entire time slice while running, its' priority is reduced (i.e. it moves down on queue).
  - **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the same priority level (e.g I/O Operation)

\* Means that the shifting down of priority level only depends on CPU time

### Example (Along Came a Short Job):

- 1) A job  $A$  enters system
- 2) Job is placed on highest Queue  $Q_2$
- 3) After time-slice (e.g. 10 ms) in  $Q_2$ ,  $A$  is placed on lower queue  $Q_1$
- 4) After time-slice in  $Q_1$ ,  $A$  is placed in lowest priority queue  $Q_0$



### • Attemp #2: The Priority Boost

- **Rule 5:** After some time period  $S$ , move all the jobs in the system to the topmost queue.
- \* This is to prevent starvation (i.e. a job never being run)

### • Attempt #3: Better Accounting (Fix of Attempt # 1)

- Is to prevent programmers from gaming (i.e tricking) the CPU so all programs get a fair share of allotment time
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (it moves down one queue).