CSC 369 Worksheet 5 Solution

August 19, 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

   ,

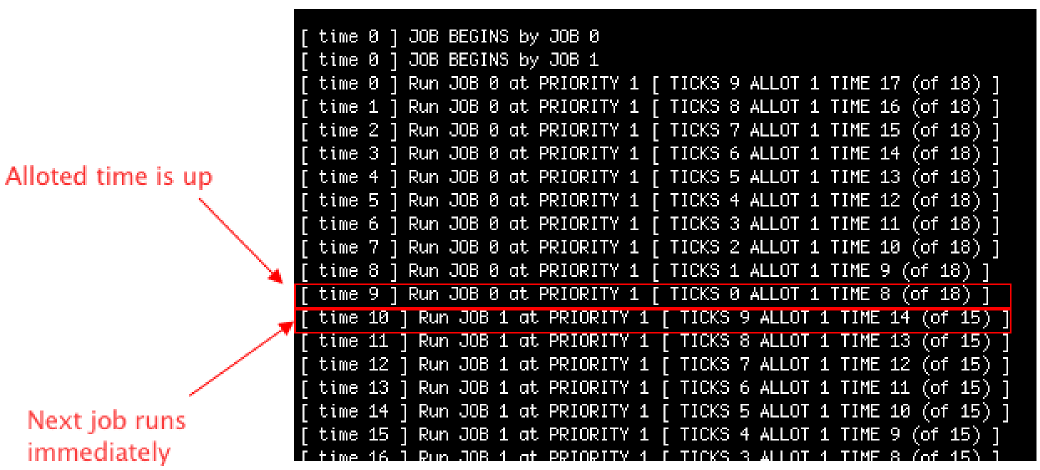
   the exeuction trace is:

   ```
   [time 0] Job begins by job 0
   [time 0] Job begins by job 1
   [time 0] Run job 0 at priority 1 [Ticks 9, Allotment 1, Time 1 (of 2)]
   [time 1] Run job 0 at priority 1 [Ticks 8, Allotment 1, Time 0 (of 2)]
   [time 2] Finished JOB 0
   [time 2] Run job 1 at priority 1 [Ticks 9, Allotment 1, Time 6 (of 7)]
   ```

```
 [time 3] Run job 1 at priority 1 [Ticks 8, Allotment 1, Time 5 (of 7)]
 [time 4] Run job 1 at priority 1 [Ticks 7, Allotment 1, Time 4 (of 7)]
 [time 5] Run job 1 at priority 1 [Ticks 6, Allotment 1, Time 3 (of 7)]
 [time 6] Run job 1 at priority 1 [Ticks 5, Allotment 1, Time 2 (of 7)]
 [time 7] Run job 1 at priority 1 [Ticks 4, Allotment 1, Time 1 (of 7)]
 [time 8] Run job 1 at priority 1 [Ticks 3, Allotment 1, Time 0 (of 7)]
 [time 9] Finished JOB 1
```

Notes

- Learned that when alloted time is up, the next job starts immediately (`./mlfq.py -s 20 -m 20 -n 2 -j 2 -M 0 -c`)

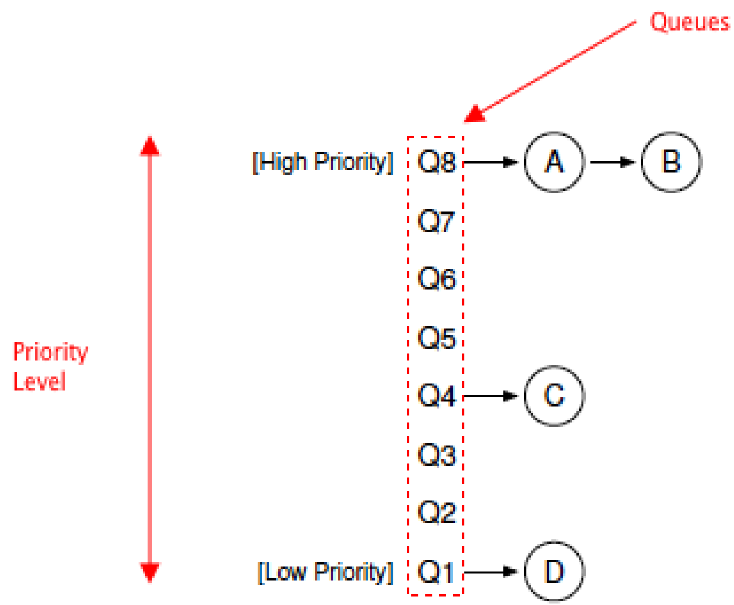


- Learned that when all jobs are at the bottom, without priority boost, jobs finishes by round robin (`./mlfq.py -s 20 -m 20 -n 2 -j 2 -M 0 -c`)

Round Robin

```
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 7 (of 18) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 6 (of 18) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 5 (of 18) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 4 (of 18) ]
[ time 24 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 3 (of 18) ]
[ time 25 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 2 (of 18) ]
[ time 26 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 1 (of 18) ]
[ time 27 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 0 (of 18) ]
[ time 28 ] FINISHED JOB 0
[ time 28 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 4 (of 15) ]
[ time 29 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 3 (of 15) ]
[ time 30 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 2 (of 15) ]
[ time 31 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 1 (of 15) ]
[ time 32 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 0 (of 15) ]
[ time 33 ] FINISHED JOB 1
```

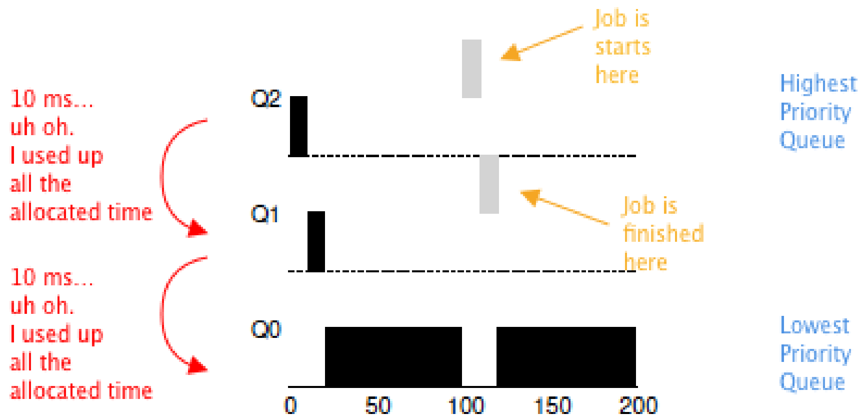
- Learned that notification and subsequent job execution happen at the same time.
- The reason why round robin doesn't occur despite Priority(A) = Priority(B) is because allotment of queue is 1 (i.e. only one job can be in a queue)
- **allotment** means the amount of something allocated to a person/object (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 Feeback 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 Priority(A) > Priority(B), A runs (B doesn't)
  - **Rule 2:** If Priority(A) = Priority(B), A & B run in RR



- **Attemp #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).

2. I need to run the scheduler (`mlfq.py`) to reproduce each of the examples in the chapter.

- Example 1: A Single Long-Running Job

  Here, the example has

  - 3 queues
  - 1 job
  - `10ms` as quantum length for queue 1
  - `10ms` as quantum length for queue 2
  - `10ms` as quantum length for queue 3
  - `200ms` as run time for job 1
  - no priority boost

  Combining together we have

```
./mlfq.py -l 0,200,0 -n 3 -j 1 -c
```

- Example 2: Along Came A Short Job

  Here, the example has

  - 3 queues
  - 2 jobs
  - `10ms` as quantum length for queue 1
  - `10ms` as quantum length for queue 2
  - `10ms` as quantum length for queue 3
  - `180ms` as run time for job 1
  - `20ms` as run time for job 2
  - 0ms as the starting time for job 1
  - 100ms as the starting time for job 2
  - no I/O operations for job 1
  - no I/O operations for job 2
  - no priority boost

  Combining together we have

  ```
  ./mlfq.py -l 0,180,0:100,20,0 -n 3 -j 2 -c
  ```

- Example 3: What About I/O

  Here, the example has

  - 3 queues
  - 2 jobs
  - `10ms` as quantum length for queue 1
  - `10ms` as quantum length for queue 2
  - `10ms` as quantum length for queue 3
  - `0ms` as the starting time for job 1
  - `185ms` as run time for job 1
  - no I/O operations for job 1
  - `50ms` as the starting time for job 2
  - `15ms` as run time for job 2
  - `1ms` as the frequency of I/O request (Start immediately)
  - `9ms` as the I/O time for job 2
  - no priority boost
  - Uses older rules (Rule 4.a, Rule 4.b)

  Combining together we have

  ```
  ./mlfq.py -l 0,185,0:50,15,1 -i 9 -n 3 -j 2 -S -c
  ```

3. I need to answer the question how would `.mlfq.py` be configured to behave just like round-robin scheduler.

   The round-robin scheduler occurs when jobs are at bottom most priority with no priority boost and I/O operations (i.e. jobs are at same priority until completion).

   So, the configuration is:

   ```
   ./mlfq.py -l 0,100,0:0,100,0 -n 1 -j 2 -c
   ```

4. I need to craft a workload with two jobs and scheduler parameters so that one job takes of the older rules and obtain 99% of the CPU over a particular time interval.

   The following configuration allows job 1 to occupy most of CPU time.

   ```
   ./mlfq.py -l 0,100,9:0,100,0 -i 1 -n 3 -j 2 -S -c
   ```

   This occurs once job 2 moves down to lower priority

5. Let a system has quantum length of `10ms` in its highest queue. Assume no I/O operations are in place.

   I need to answer how often job needs to be bosted back to the highest priority level so job gets at least 5% of CPU.

   Given `500ms` turn around time, the job needs to be boosted every `100ms` to get at least 5% CPU time.

   Notes

   - I need help from professor regarding CPU time calculation. How would we know if a process is occupying 99% of cpu time? Or, 5% of CPU time? Is there a formula behind the CPU calculation?

6. I need to write the effect of `-I` flag on scheduling once I/O finishes.

   The `-I` flag allows a process after an I/O operation to be placed at the front of current queue.

   With this flag, the job 1 in question 4 can truly achieve 99% CPU time.

   The following is the command used to achieve 99% of CPU time.

   ```
   ./mlfq.py -l 0,100,9:0,100,0 -i 1 -n 3 -j 2 -I -S -c
   ```