1 Homework Questions

MLFQ Summary

Rule 1: If Priority(A)>Priority(B), A runs (B doesn't)
Rule 2: If Priority(A)=Priority(B), A & B run in round-robin fashion using the time slice (quantum length) of the given queue.
Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue)
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 (i.e., it moves down one queue).
Rule 5: After some time period S, move all the jobs in the system to the topmost queue.

1. **Run a few randomly-generated problems with just two jobs and two queues; compute the MLFQ execution trace for each. Make your life easuer by limiting the length of each job and turning off I/0s.**

```
python mlfq.py -j 2 -n 2 -m 10 -M 0 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS quantum length for queue  1 is  10
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS stayAfterIO False
OPTIONS iobump False

For each job, three defining characteristics are given:
startTime : at what time does the job enter the system
runTime   : the total CPU time needed by the job to finish
ioFreq    : every ioFreq time units, the job issues an I/O
            (the I/O takes ioTime units to complete)

Job List:
Job  0: startTime   0 - runTime   8 - ioFreq   0
Job  1: startTime   0 - runTime   4 - ioFreq   0

Execution Trace:

[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 7 (of 8) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 6 (of 8) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 5 (of 8) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 4 (of 8) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 3 (of 8) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 2 (of 8) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 1 (of 8) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 0 (of 8) ]
[ time 8 ] FINISHED JOB 0
[ time 8 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 3 (of 4) ]
[ time 9 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 2 (of 4) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 1 (of 4) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 0 (of 4) ]
[ time 12 ] FINISHED JOB 1

Final statistics:
Job  0: startTime   0 - response   0 - turnaround   8
Job  1: startTime   0 - response   8 - turnaround  12

Avg  1: startTime n/a - response 4.00 - turnaround 10.00
```

2. **How would you run the scheduler to reproduce each of the examples in the chapter?** Figure 8.2

```
python mlfq.py -n 3 -q 10 -M 0 -l 0,200,0 -c
```

Figure 8.3

```
python mlfq.py -n 3 -q 10 -l 0,180,0:100,20,0 -c
```

Figure 8.4

```
python mlfq.py -n 3 -q 10 -i 4 -S -l 0,170,0:50,30,1 -c
```

Figure 8.5 (Right)

```
python mlfq.py -n 3 -q 10 -B 50 -S -i 5 -l 0,120,0:100,40,5:100,40,5 -c
```

Figure 8.6 (Right, for Left just add -S)

```
python mlfq.py -l 0,200,0:30,200,9 -q 10 -n 3 -i 1 -c
```

Figure 8.7

```
python mlfq.py -n 3 -Q 10,20,40 -a 2 -l 0,150,0:0,150,0 -c
```

3. **How would you configure the scheduler parameters to behave just like a round-robin scheduler?**

   You could just have one queue so that all processes are at the same priority (-n 1).

4. **Craft a workload with two jobs and scheduler parameters so that one job takes advantage of the older rules 4a and 4b to game the scheduler and obtain 99% of the CPU over a particular time interval.**

   ```
   python mlfq.py -n 2 -q 100 -S -i 1 -l 100,1000,99:0,1000,0 -c
   ```

5. **Given a system with a quantum length of 10ms in its highest queue, how often would you have to boost jobs back to the highest priority level (with the -B flag) in order to guarantee that a single long-running (and potentially-starving) job gets at least 5% of the CPU?**

   5% $= \frac{1}{20}$ of CPU. Since each time slice is 10ms, the boost has to be every 200ms (20 time slices).

6. **One question that arises in scheduling is which end of a queue to add a job that just finished I/O; the -I flag changes this behavior for this schedulating simulator. Play around with some workloads and see if you can see the effect of this flag.**

   ```
   python mlfq.py -l 0,50,0:0,50,5 -n 1 -q 10 -i 5 -c
   python mlfq.py -l 0,50,0:0,50,5 -n 1 -q 10 -i 5 -I -c
   ```