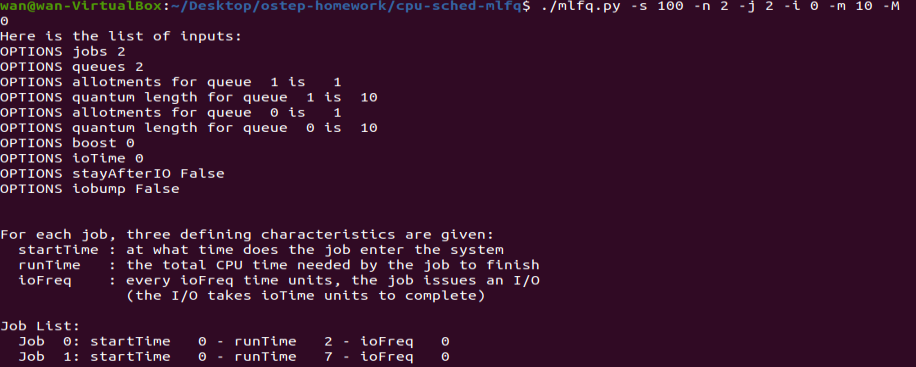
**Homework (Simulation) Wan Huzaifah bin Wan Azhar**

This program, mlfq.py, allows you to see how the MLFQ schedulerpresented in this chapter behaves. See the README for details

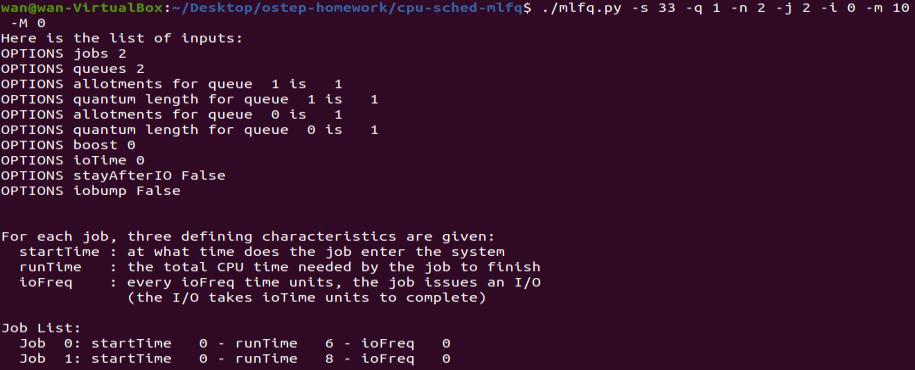
**Answer:**

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



Execution trace:

Time 0: Job begin Job 0  
Time 0: Job begin Job 1  
Time 1: Job 0 run at priority 1 – 9 time left  
Time 2: Job 0 run at priority 1 – 8 time left  
Time 2: Job 0 finished  
Time 3: Job 1 run at priority 1 – 9 time left  
Time 4: Job 1 run at priority 1 – 8 time left  
Time 5: Job 1 run at priority 1 – 7 time left  
Time 6: Job 1 run at priority 1 – 6 time left  
Time 7: Job 1 run at priority 1 – 5 time left  
Time 8: Job 1 run at priority 1 – 4 time left  
Time 9: Job 1 run at priority 1 – 3 time left  
Time 9s: Job 1 finished



Execution trace:

Time 0: Job begin Job 0  
Time 0: Job begin Job 1  
Time 1: Job 0 run at priority 1  
Time 2: Job 1 run at priority 1  
Time 2: Job 0 run at priority 2  
Time 3: Job 1 run at priority 2  
Time 4: Job 0 run at priority 2  
Time 5: Job 1 run at priority 2  
Time 6: Job 0 run at priority 2  
Time 7: Job 1 run at priority 2  
Time 8: Job 0 run at priority 2  
Time 9: Job 1 run at priority 2  
Time 10: Job 0 run at priority 2  
Time 10: Job 0 finished  
Time 11: Job 1 run at priority 2  
Time 12: Job 1 run at priority 2  
Time 13: Job 1 run at priority 2  
Time 14: Job 1 finished

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

Figure 8.2: python mlfq.py -l 0,200,0 -q 10 -B 0

Figure 8.3: python mlfq.py -l 0,200,0:100,20,0 -q 10 -B 0

Figure 8.4: python mlfq.py -l 0,175,0:50,25,1 -q 10 -S

Figure 8.5a: python mlfq.py -l 0,175,0:50,25,1:50,25,1 -q 10 -S -i 1 -B 0

Figure 8.5b: python mlfq.py -l 0,175,0:50,25,1:50,25,1 -q 10 -S -i 1 -B 50

Figure 8.6a: python mlfq.py -l 0,175,0:80,100,9 -q 10 -S -i 1

Figure 8.6b: python mlfq.py -l 0,175,0:80,80,9 -q 10 -i 1

Figure 8.7: python mlfq.py -l 0,200,0:0,200,0 -q 10 -Q 10,20,40

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

python mlfq.py -s 44 -q 10 -i 0 -M 0

With the assumptions that IO is not configured, set the time slice to 10ms. Average response is 10ms.

**4. Craft a workload with two jobs and scheduler parameters so that one job takes advantage of the older Rules 4a and 4b (turned on with the -S flag) to game the scheduler and obtain 99% of the CPU over particular time interval.**

Python mlfq.py -l 0,200,0:10,100,9 -S -q 10 -i 1 -B 0

This will make it so that Job 1 will take over the CPU starting at time 10 to completion. It will start light I/O request, which will finish in 1ms so that its priority is always at the highest.

**5. Given a system with 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?**

Python mlfq-py -l 0,200,0:10,100,9 -S -Q 10,20,40 -I 1 -B 64

I don’t think there is specific Boost number to guarantee 5% of the CPU. But in this question, In order to get at least 5% of CPU over 100 Jobs 1 (that games the scheduler), the Boost has to be around 2/3 of the Jobs. As such 2/3 of 100 is 64. 64 is a good number. If the Job 1 has runtime of 200, the boost of 64 will guarantee that Job 0 will run three times: 64, 128, 192.

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

The -I flag prioritize the job that finished IO first. So a Job that finished IO will start first over job that has the same priority. Even though another job without IO start first, as the Job that has IO is done, the Job that finish IO will override the job without IO. It is no longer round-robin.