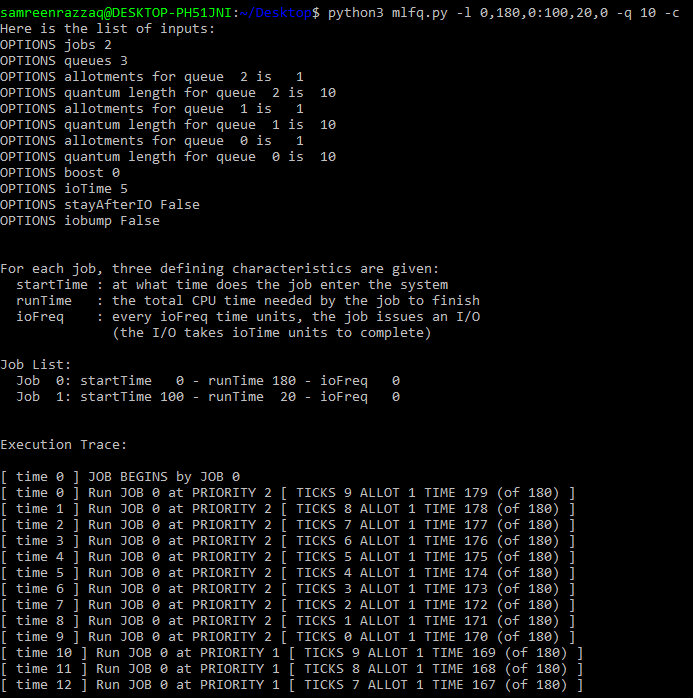
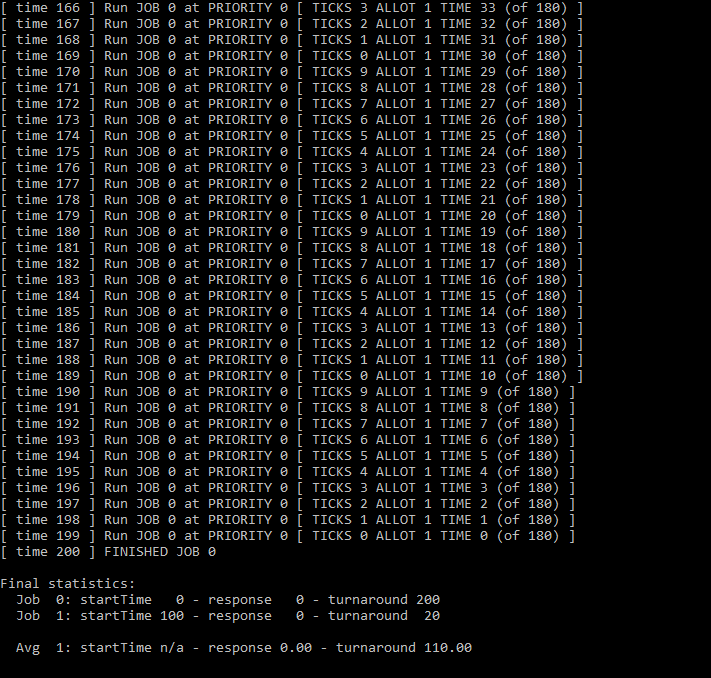


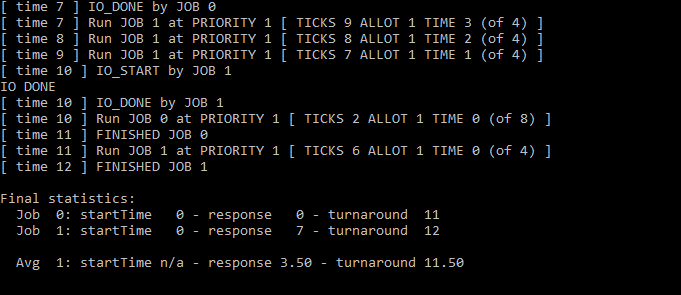
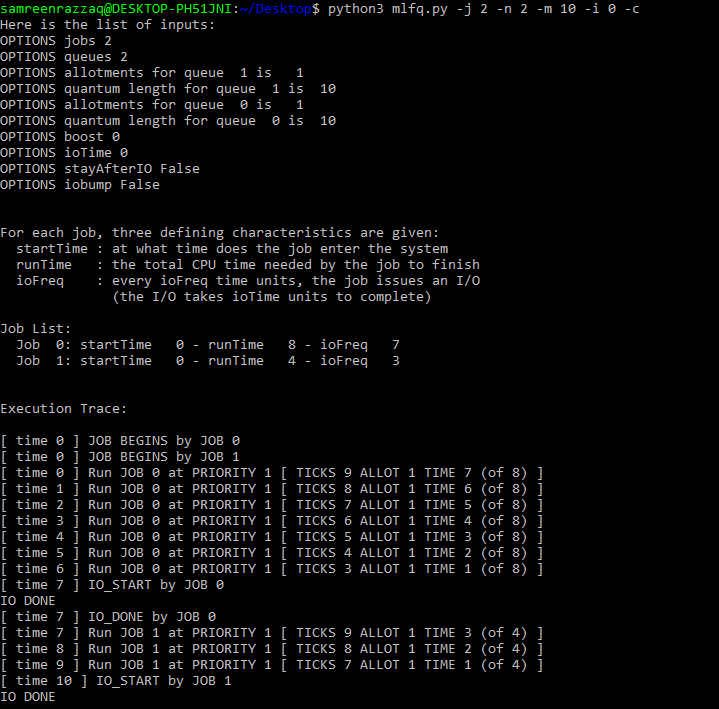
**Lab Tasks:**

Compute the execution trace for the given workloads. Also compute the response and turnaround times for each of the jobs. Use the -c flag to get the exact results when you are finished.

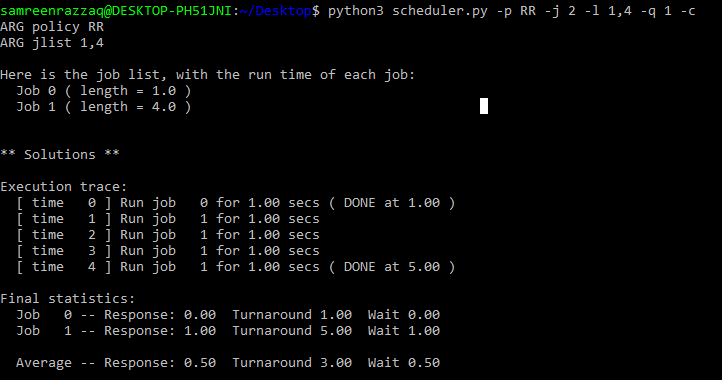
1. Recreate the example in Figure 8.3 you would specify a job list as follows: prompt> ./mlfq.py --jlist 0,180,0:100,20,0 -q 10

 Running the simulator in this way creates a three-level MLFQ, with each level having a 10-ms time slice. Two jobs are created: Job 0 which starts at time 0, runs for 180 ms total, and never issues an I/O; Job 1 starts at 100 ms, needs only 20 ms of CPU time to complete, and also never issues I/Os.

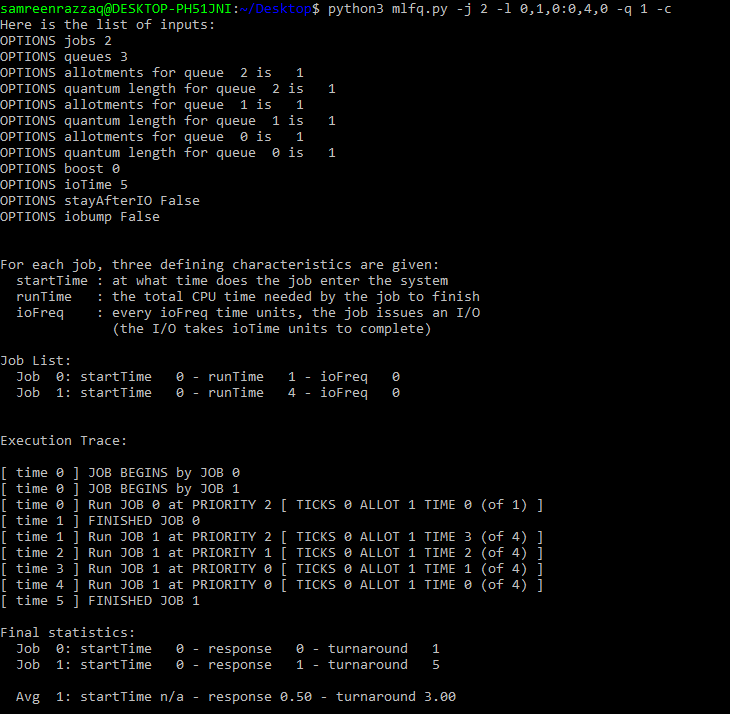


1. Run a few randomly-generated problems with just two jobs and two queues; compute the MLFQ execution trace for each. Make your life easier by limiting the length of each job and turning off I/Os.
2. How would you configure the scheduler parameters to behave just like a round-robin scheduler?

**By using scheduler.py:**

****

**By using mlfq.py:**

**** Turnaround and Response time is same in both.

1. 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?

We assume that there is inverse relation between boost time and turnaround. With the decrease of boot time, the efficiency become better. We check this by giving any two lengths:

