Operating Systems Assignment 4 (CLO 1, PLO1, Total Marks 30)

The program, mlfq.py, allows you to see how the MLFQ scheduler presented in this chapter behaves. See the README for details.
Questions

1. [2] Run 2 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. [6] How would you run the scheduler to reproduce the following examples: Example 2 (Figure 8.3 left), Example 3 (Figure 8.3 right), Boosting (Figure 8.4 right)

3. [5] How would you configure the scheduler parameters to behave just like a round-robin scheduler?

4. [7] 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 a particular time interval.

5. [5] Given a system with a quantum length of 10 ms 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 longrunning (and potentially-starving) job gets at least 5% of the CPU?

6. [5] 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 scheduling simulator. Play around with some workloads and see if you can see the effect of this flag.

(Solution)

(1)

Terminal Command:
./mlfq.py --jlist 0,3,0:2,6,0 -q 2

- Two jobs, and two queues.
- Arrival of the first job is at 0, compute time is 3, and there is no I/O request.
- Arrival of the second job is at 2, compute time is 6, and there is no I/O request.

Terminal Output:

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/009_HA4/ostep-homework/cpu-sched-mlfq$ ./mlfq.py --jlist 0,3,0:2,6,0 -q 2
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 3
OPTIONS allotments for queue  2 is   1
OPTIONS quantum length for queue  2 is   2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is   2
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is   2
OPTIONS boost 0
OPTIONS ioTime 5
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   3 - ioFreq   0
  Job  1: startTime   2 - runTime   6 - ioFreq   0

Compute the execution trace for the given workloads.
If you would like, 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.
```

Terminal Command:
./mlfq.py --jlist 0,3,0:2,6,0 -q 2 -c

Terminal Output:

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/009_HA4/ostep-homework/cpu-sched-mlfq$ ./mlfq.py --jlist 0,3,0:2,6,0 -q 2 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 3
OPTIONS allotments for queue  2 is   1
OPTIONS quantum length for queue  2 is   2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is   2
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is   2
OPTIONS boost 0
OPTIONS ioTime 5
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   3 - ioFreq   0
  Job  1: startTime   2 - runTime   6 - ioFreq   0


Execution Trace:

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

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround   5
  Job  1: startTime   2 - response   0 - turnaround   7

  Avg  1: startTime n/a - response 0.00 - turnaround 6.00
```

(2)

Terminal Command:
./mlfq.py --jlist 0,180,0:100,20,0 -q 10

- Picked from readme file.

Terminal Output:

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/009_HA4/ostep-homework/cpu-sched-mlfq$ ./mlfq.py --jlist 0,180,0:100,20,0 -q 10
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 3
OPTIONS allotments for queue  2 is   1
OPTIONS quantum length for queue  2 is  10
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS ioTime 5
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 180 - ioFreq   0
  Job  1: startTime 100 - runTime  20 - ioFreq   0

Compute the execution trace for the given workloads.
If you would like, 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.
```

Terminal Command:
./mlfq.py --jlist 0,180,0:100,20,0 -q 10 -B 100

- Adding boost after every 100ms.

Terminal Output:

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/009_HA4/ostep-homework/cpu-sched-mlfq$ ./mlfq.py --jlist 0,180,0:100,20,0 -q 10 -B 100
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 3
OPTIONS allotments for queue  2 is   1
OPTIONS quantum length for queue  2 is  10
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 100
OPTIONS ioTime 5
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 180 - ioFreq   0
  Job  1: startTime 100 - runTime  20 - ioFreq   0

Compute the execution trace for the given workloads.
If you would like, 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.
```

(3)

Let us think of a simple scenario in which there are 2 queues, quanta of queue 1 is 1, quanta of queue 2 is 2, allotment of queue 1 is 1, and allotment of queue 2 is 2. System boasts after 10 time units.

Now there are two jobs: Both jobs arrive at the same time, and their computation time is 20.

./mlfq.py --jlist 0,20,0:0,20,0 -n 2 -Q 1,2 -A 1,2 -B 10

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/009_HA4/ostep-homework/cpu-sched-mlfq$ ./mlfq.py --jlist 0,20,0:0,20,0 -n 2 -Q 1,2 -A 1,2 -B 10
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is   1
OPTIONS allotments for queue  0 is   2
OPTIONS quantum length for queue  0 is   2
OPTIONS boost 10
OPTIONS ioTime 5
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  20 - ioFreq   0
  Job  1: startTime   0 - runTime  20 - ioFreq   0

Compute the execution trace for the given workloads.
If you would like, 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.
```

After adding -c flag

```
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 0 ALLOT 1 TIME 19 (of 20) ]
[ time 1 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 19 (of 20) ]
[ time 2 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 18 (of 20) ]
[ time 3 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 17 (of 20) ]
[ time 4 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 18 (of 20) ]
[ time 5 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 17 (of 20) ]
[ time 6 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 16 (of 20) ]
[ time 7 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 15 (of 20) ]
[ time 8 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 16 (of 20) ]
[ time 9 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 15 (of 20) ]
[ time 10 ] BOOST ( every 10 )
[ time 10 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 14 (of 20) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 14 (of 20) ]
[ time 12 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 13 (of 20) ]
[ time 13 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 12 (of 20) ]
[ time 14 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 13 (of 20) ]
[ time 15 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 12 (of 20) ]
[ time 16 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 11 (of 20) ]
[ time 17 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 10 (of 20) ]
[ time 18 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 11 (of 20) ]
[ time 19 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 10 (of 20) ]
[ time 20 ] BOOST ( every 10 )
[ time 20 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 9 (of 20) ]
[ time 21 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 9 (of 20) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 8 (of 20) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 7 (of 20) ]
[ time 24 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 8 (of 20) ]
[ time 25 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 7 (of 20) ]
[ time 26 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 6 (of 20) ]
[ time 27 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 5 (of 20) ]
[ time 28 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 6 (of 20) ]
[ time 29 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 5 (of 20) ]
[ time 30 ] BOOST ( every 10 )
[ time 30 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 4 (of 20) ]
[ time 31 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 4 (of 20) ]
[ time 32 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 3 (of 20) ]
[ time 33 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 2 (of 20) ]
[ time 34 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 2 TIME 3 (of 20) ]
[ time 35 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 2 TIME 2 (of 20) ]
[ time 36 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 1 (of 20) ]
[ time 37 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 0 (of 20) ]
[ time 38 ] FINISHED JOB 0
[ time 38 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 1 (of 20) ]
[ time 39 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 0 (of 20) ]
[ time 40 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  38
  Job  1: startTime   0 - response   1 - turnaround  40

  Avg  1: startTime n/a - response 0.50 - turnaround 39.00
```

(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 a particular time interval.

Let us think of a scenario in which there are 2 queues, the quanta of queue 1 is 100, the quanta of queue 2 is 100, the allotment of queue 1 is 1, and allotment of queue 2 is 1 as well. I/O time is 1 unit.
Now trick task 1 will play is that it will issue an I/O request after 99 compute units.

Terminal Command:
./mlfq.py --jlist 0,125,0:0,400,99 -n 2 -Q 100,100 -A 1,1 -i 1 -c -S

- Job 0 has 125 time units computation, and I/O requests are never issued.
- Job 1 has 400 time units computation, and issues an I/O request after every 99 time units.

What do I want to observe as a result of the above terminal command?

- Job 0 does not issue I/O. Hence downgrades to lower priority queue after 100 time units.
- Job 1 takes over runs for 99 time units, and issues an I/O request.
- Then Job 0 runs for 1 cycle which is on a low priority queue.
- Then Job 1 comes back, keeps going for 99 time units, and then issues I/O requests.
- Then Job 0 runs for 1 cycle which is on a low priority queue.
- This keeps on repeating till Job 1 finishes its jobs. This is how 99% of the compute time is hogged by Job 1.

Terminal Output:
Here, We can see that Job 1 finishes first, and Job 0 despite low runtime finishes last.

```
[ time 504 ] FINISHED JOB 1
[ time 504 ] Run JOB 0 at PRIORITY 0 [ TICKS 95 ALLOT 1 TIME 20 (of 125) ]
[ time 505 ] Run JOB 0 at PRIORITY 0 [ TICKS 94 ALLOT 1 TIME 19 (of 125) ]
[ time 506 ] Run JOB 0 at PRIORITY 0 [ TICKS 93 ALLOT 1 TIME 18 (of 125) ]
[ time 507 ] Run JOB 0 at PRIORITY 0 [ TICKS 92 ALLOT 1 TIME 17 (of 125) ]
[ time 508 ] Run JOB 0 at PRIORITY 0 [ TICKS 91 ALLOT 1 TIME 16 (of 125) ]
[ time 509 ] Run JOB 0 at PRIORITY 0 [ TICKS 90 ALLOT 1 TIME 15 (of 125) ]
[ time 510 ] Run JOB 0 at PRIORITY 0 [ TICKS 89 ALLOT 1 TIME 14 (of 125) ]
[ time 511 ] Run JOB 0 at PRIORITY 0 [ TICKS 88 ALLOT 1 TIME 13 (of 125) ]
[ time 512 ] Run JOB 0 at PRIORITY 0 [ TICKS 87 ALLOT 1 TIME 12 (of 125) ]
[ time 513 ] Run JOB 0 at PRIORITY 0 [ TICKS 86 ALLOT 1 TIME 11 (of 125) ]
[ time 514 ] Run JOB 0 at PRIORITY 0 [ TICKS 85 ALLOT 1 TIME 10 (of 125) ]
[ time 515 ] Run JOB 0 at PRIORITY 0 [ TICKS 84 ALLOT 1 TIME 9 (of 125) ]
[ time 516 ] Run JOB 0 at PRIORITY 0 [ TICKS 83 ALLOT 1 TIME 8 (of 125) ]
[ time 517 ] Run JOB 0 at PRIORITY 0 [ TICKS 82 ALLOT 1 TIME 7 (of 125) ]
[ time 518 ] Run JOB 0 at PRIORITY 0 [ TICKS 81 ALLOT 1 TIME 6 (of 125) ]
[ time 519 ] Run JOB 0 at PRIORITY 0 [ TICKS 80 ALLOT 1 TIME 5 (of 125) ]
[ time 520 ] Run JOB 0 at PRIORITY 0 [ TICKS 79 ALLOT 1 TIME 4 (of 125) ]
[ time 521 ] Run JOB 0 at PRIORITY 0 [ TICKS 78 ALLOT 1 TIME 3 (of 125) ]
[ time 522 ] Run JOB 0 at PRIORITY 0 [ TICKS 77 ALLOT 1 TIME 2 (of 125) ]
[ time 523 ] Run JOB 0 at PRIORITY 0 [ TICKS 76 ALLOT 1 TIME 1 (of 125) ]
[ time 524 ] Run JOB 0 at PRIORITY 0 [ TICKS 75 ALLOT 1 TIME 0 (of 125) ]
[ time 525 ] FINISHED JOB 0

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround 525
  Job  1: startTime   0 - response 100 - turnaround 504
```

(5)

Given a system with a quantum length of 10 ms 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 longrunning (and potentially-starving) job gets at least 5% of the CPU?

10 ms is the guaranteed time that a task would run given a quantum length of 10 ms in the highest priority queue.
Boast period should be 10 ms / (5 %) which is **200 ms**.

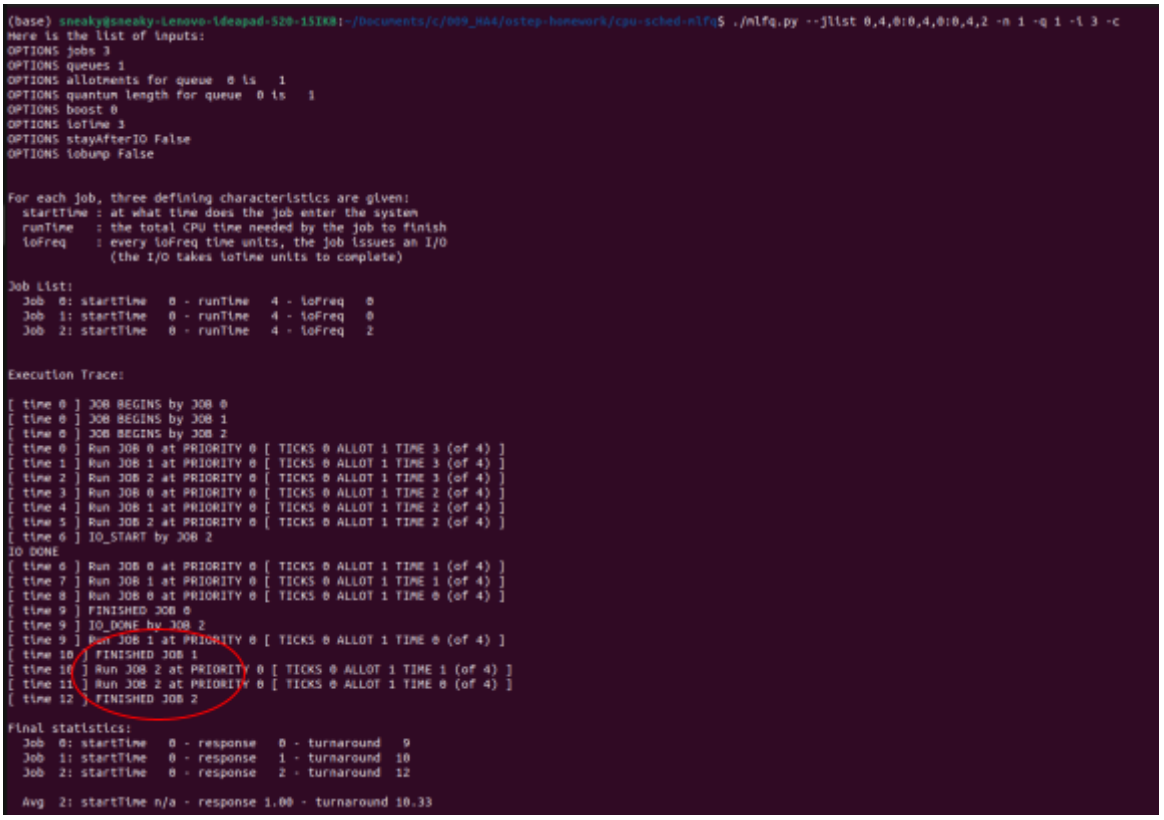
(6)

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 scheduling simulator. Play around with some workloads and see if you can see the effect of this flag.

-I Flag defines the queue scheduling behavior on I/O request response to the system. When we add -I flag at the end of the command then the I/O request initiator task is catered next immediately once the I/O request has been processed.

Terminal Command:
./mlfq.py --jlist 0,4,2:0,4,0:0,4,0 -n 1 -q 1 -i 3 -c

Terminal Output:



Terminal Command:
./mlfq.py --jlist 0,4,2:0,4,0:0,4,0 -n 1 -q 1 -i 3 -c -I

Terminal Output: