Chapter 8 Simulator

1.
python .\mlfq.py -M 0 -j 2 -n 2 -A 2,3 -m 50 -c with different seeds.
Also modified -A to see different behaviors under different allotment time at each level.

2.
Example 1 is a single long running job with 3 queues:
-M 0 -j 1 -n 3 -A 1,1,1

Example 2 is a CPU intensive long running job, a short job, with 3 queues:
-j 2 -n 3 -I 0,100,0:40,20,0

Example 3 adds I/O, so set -S stay to keep the priority of a process in the same level.
-j 2 -n 3 -I 0,100,0:40,50,5 -S stay

Example: one process games the CPU by issuing very frequently a trivial IO
-j 2 -n 3 -I 0,100,0:40,50,5 -i 1 -S stay

Example: with boost by specifying -B
-j 2 -n 3 -I 0,100,0:40,50,5 -i 1 -S stay -B 35

3.
For example, 3 similar programs without IO.
-j 3 -n 3 -I 0,30,0:0,30,0:0,30,0

4. In this case, the second job starting at 30ms, lasting 50ms will use approximately 99% of cpu during its run time.
-j 2 -n 3 -I 0,100,0:30,50,5 -i 1 -S stay

5.
Assume the worst case that a single long process is starving at the bottom of the queues because too many interactive jobs are running in the higher levels. Then at least we need to boost every 20 time slices (-B 200), which is 200ms.

6.
If I is specified, the process that finishes IO will be returned to the front of the queue. This makes the sneaky process (that stays in high level with frequent trivial IO) easier to gain more cpu, even when boost is used.
On the other hand, if it's a normal IO intensive program, this approach may improve the performance of the system, because we quickly release the IOs of these IO intensive program so these programs won't wait for long time, and the cpu can also switch to other processes during this time.

Chapter 9

1. Compute the solutions for simulations with 3 jobs and random seeds of 1, 2, and 3.

The modulo method divide the large random number by total number of tickets, thus the remainder are between 0 to (total tickets -1). For example in seed 1, job 0 has 84 tickets, job 1 has 25, job 2 has 44, so if winning number is 84 then it belongs to job 1, as job 0 only wins when winning number is between 0-83.

After a job is done, its tickets should be removed from the total number of tickets.

2. Now run with two specific jobs: each of length 10, but one (job 0) with just 1 ticket and the other (job 1) with 100 (e.g., -l 10:1,10:100). What happens when the number of tickets is so imbalanced? Will job 0 ever run before job 1 completes? How often? In general, what does such a ticket imbalance do to the behavior of lottery scheduling?

Job 0 can barely run before job 1 completes, because every time slice job 1 have 100 times more chance to run than job 0.

A drastic ticket imbalance can lead to the processes with most tickets to monopolize the cpu. If they happen to be long processes, the other processes with least tickets will have very bad response time and completion time.
It might be better to consider give more tickets to short, interactive jobs.

3. When running with two jobs of length 100 and equal ticket allocations of 100 (-l 100:100,100:100), how unfair is the scheduler? Run with some different random seeds to determine the (probabilistic) answer; let unfairness be determined by how much earlier one job finishes than the other.

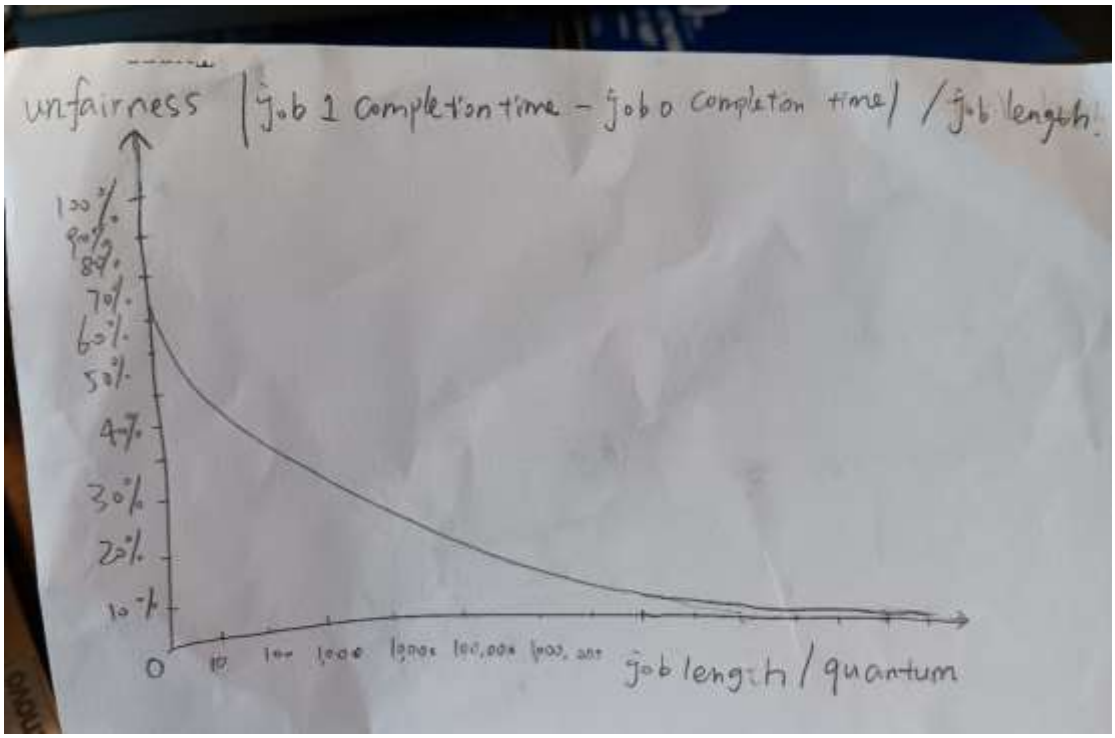
Unfairness = | Completion time job 0 - Completion time job 1|
Perfectly fair means Unfairness = 0
Totally unfair means Unfairness = 100

For seeds 1 to 10, the unfairness are 4,10,4,1,19,7,15,9,8,3
Average unfairness = 8
One job completes 8% ahead of another with same length and tickets.

4. How does your answer to the previous question change as the quantum size (-q) gets larger?
The larger the quantum, the less times of winning ticket selection, therefore the more unfairness in lottery scheduler.

5. Can you make a version of the graph that is found in the chapter? What else would be worth exploring? How would the graph look with a stride scheduler?

(1)



(2)

A stride scheduler is completely fair, as we use counter (pass value) to increment the stride values every time one process runs to eliminate randomness. So its unfairness will remain 0 no matter the job length and quantum. The graph should be a horizontal line.