CS5600 Homework 3
Zongwang Wang

Chapter 8

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.

```
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  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  17 - ioFreq   0
  Job  1: startTime   0 - runTime   8 - 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 16 (of 17) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 15 (of 17) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 14 (of 17) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 13 (of 17) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 12 (of 17) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 11 (of 17) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 10 (of 17) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 9 (of 17) ]
[ time 8 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 8 (of 17) ]
[ time 9 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 7 (of 17) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 7 (of 8) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 6 (of 8) ]
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 5 (of 8) ]
[ time 13 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 4 (of 8) ]
[ time 14 ] Run JOB 1 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 3 (of 8) ]
[ time 15 ] Run JOB 1 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 2 (of 8) ]
[ time 16 ] Run JOB 1 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 1 (of 8) ]
[ time 17 ] Run JOB 1 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 0 (of 8) ]
[ time 18 ] FINISHED JOB 1
[ time 18 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 6 (of 17) ]
[ time 19 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 5 (of 17) ]
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 4 (of 17) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 3 (of 17) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 2 (of 17) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 1 (of 17) ]
[ time 24 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 0 (of 17) ]
[ time 25 ] FINISHED JOB 0
Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  25
  Job  1: startTime   0 - response  10 - turnaround  18
  Avg  1: startTime n/a - response 5.00 - turnaround 21.50
```

CS5600 Homework 3
Zongwang Wang

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

Example 1: A Single Long-Running Job (Figure 8.2)

./mlfq.py -n 3 -q 10 -l 0,200,0 -c

Example 2: Along Came A Short Job (Figure 8.3, left)
./mlfq.py -n 3 -q 10 -l 0,180,0:100,20,0 -c

Example 3: What About I/O? (Figure 8.3, right)
./mlfq.py -n 3 -q 10 -l 0,180,0:100,20,1 -i 5 -c
- Job B now does I/O every 1ms
- -i 5: I/O takes 5ms

Example with Priority Boost (Figure 8.4)
Without boost:
./mlfq.py -n 3 -q 10 -l 0,180,0:100,20,1:100,20,1 -i 5 -c

With boost:
./mlfq.py -n 3 -q 10 -l 0,180,0:100,20,1:100,20,1 -i 5 -B 100 -c
-`-B 100: Boost every 100ms

Gaming Example (Figure 8.5)
Without protection:
./mlfq.py -n 3 -q 10 -l 0,200,0:0,100,9 -i 1 -S -c
- -S: Uses old rules 4a/4b (stay at same priority after I/O)
- Job games by doing I/O every 9ms

With protection:
./mlfq.py -n 3 -q 10 -l 0,200,0:0,100,9 -i 1 -c
- Without -S to proper accounting prevents gaming

Example with Different Time Slices (Figure 8.6)
./mlfq.py -n 3 -Q 10,20,40 -l 0,200,0:50,50,0 -c
- -Q 10,20,40: Different quantum lengths per queue

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

./mlfq.py -n 1 -q 10 -M 0 -c

Here we have just one queue with the time quantum as 10ms (for example), this is essentially a RR as the priority of the task will not change, and processes will switch every 10ms.

CS5600 Homework 3
Zongwang Wang

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.

./mlfq.py -n 3 -q 100 -l 0,1000,0:0,1000,99 -i 1 -S -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 100
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is 100
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is 100
OPTIONS boost 0
OPTIONS ioTime 1
OPTIONS stayAfterIO True
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 1000 - ioFreq   0
  Job  1: startTime   0 - runTime 1000 - ioFreq  99
Final statistics:
  Job  0: startTime   0 - response   0 - turnaround 2000
  Job  1: startTime   0 - response 100 - turnaround 1110
```

Here, Job 1 runs for 99ms out of every 100ms cycle, achieving 99% CPU utilization while keeping Job 0 starved in the lower queues.

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?

./mlfq.py -n 3 -q 10 -l 0,1000,0:0,1000,1 -i 5 -B 200 -c
With -B 200, Job 0 is guaranteed to get boosted every 200ms and run for at least 10ms at the highest priority, ensuring it gets 10ms/200ms = 5% of the CPU minimum.

CS5600 Homework 3
Zongwang Wang

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.

The -I flag controls where a job returning from I/O gets placed in its current priority queue - at the front (head) or back (tail). This small detail can significantly impact fairness and response times.

Using two example below:
Default behavior (add to tail):
bash./mlfq.py -n 1 -q 10 -l 0,100,5:0,100,5:0,100,5 -i 5 -c

With -I flag (add to front):
bash./mlfq.py -n 1 -q 10 -l 0,100,5:0,100,5:0,100,5 -i 5 -I -c
This creates 3 jobs that all do I/O every 5ms. The difference:

Without -I: Jobs returning from I/O go to the back of the queue, creating fair round-robin
With -I: Jobs returning from I/O jump to the front, potentially allowing one job to monopolize CPU

Chapter 9

1. Compute the solutions for simulations with 3 jobs and random seeds of 1, 2, and 3.
./lottery.py -j 3 -s 1

```
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 1
Here is the job list, with the run time of each job:
  Job 0 ( length = 1, tickets = 84 )
  Job 1 ( length = 7, tickets = 25 )
  Job 2 ( length = 4, tickets = 44 )
** Solutions **
Random 651593 -> Winning ticket 119 (of 153) -> Run 2
  Jobs:
 (  job:0 timeleft:1 tix:84 )  (  job:1 timeleft:7 tix:25 )  (* job:2 timeleft:4 tix:44 )
Random 788724 -> Winning ticket 9 (of 153) -> Run 0
  Jobs:
 (* job:0 timeleft:1 tix:84 )  (  job:1 timeleft:7 tix:25 )  (  job:2 timeleft:3 tix:44 )
--> JOB 0 DONE at time 2
Random 93859 -> Winning ticket 19 (of 69) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:7 tix:25 )  (  job:2 timeleft:3 tix:44 )
Random 28347 -> Winning ticket 57 (of 69) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:6 tix:25 )  (* job:2 timeleft:3 tix:44 )
Random 835765 -> Winning ticket 37 (of 69) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:6 tix:25 )  (* job:2 timeleft:2 tix:44 )
Random 432767 -> Winning ticket 68 (of 69) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:6 tix:25 )  (* job:2 timeleft:1 tix:44 )
--> JOB 2 DONE at time 6
```

CS5600 Homework 3
Zongwang Wang

Random 762280 -> Winning ticket 5 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:6 tix:25) (job:2 timeleft:0 tix:---)
Random 2106 -> Winning ticket 6 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:5 tix:25) (job:2 timeleft:0 tix:---)
Random 445387 -> Winning ticket 12 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:4 tix:25) (job:2 timeleft:0 tix:---)
Random 721540 -> Winning ticket 15 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:3 tix:25) (job:2 timeleft:0 tix:---)
Random 228762 -> Winning ticket 12 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:2 tix:25) (job:2 timeleft:0 tix:---)
Random 945271 -> Winning ticket 21 (of 25) -> Run 1
  Jobs:
 (job:0 timeleft:0 tix:---) (* job:1 timeleft:1 tix:25) (job:2 timeleft:0 tix:---)
--> JOB 1 DONE at time 12

./lottery.py -j 3 -s 2

ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 2
Here is the job list, with the run time of each job:
  Job 0 (length = 9, tickets = 94)
  Job 1 (length = 8, tickets = 73)
  Job 2 (length = 6, tickets = 30)
** Solutions **
Random 605944 -> Winning ticket 169 (of 197) -> Run 2
  Jobs:
 (job:0 timeleft:9 tix:94) (job:1 timeleft:8 tix:73) (* job:2 timeleft:6 tix:30)
Random 606802 -> Winning ticket 42 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:9 tix:94) (job:1 timeleft:8 tix:73) (job:2 timeleft:5 tix:30)
Random 581204 -> Winning ticket 54 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:8 tix:94) (job:1 timeleft:8 tix:73) (job:2 timeleft:5 tix:30)
Random 158383 -> Winning ticket 192 (of 197) -> Run 2
  Jobs:
 (job:0 timeleft:7 tix:94) (job:1 timeleft:8 tix:73) (* job:2 timeleft:5 tix:30)
Random 430670 -> Winning ticket 28 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:7 tix:94) (job:1 timeleft:8 tix:73) (job:2 timeleft:4 tix:30)
Random 393532 -> Winning ticket 123 (of 197) -> Run 1
  Jobs:
 (job:0 timeleft:6 tix:94) (* job:1 timeleft:8 tix:73) (job:2 timeleft:4 tix:30)
Random 723012 -> Winning ticket 22 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:6 tix:94) (job:1 timeleft:7 tix:73) (job:2 timeleft:4 tix:30)
Random 994820 -> Winning ticket 167 (of 197) -> Run 2
  Jobs:
 (job:0 timeleft:5 tix:94) (job:1 timeleft:7 tix:73) (* job:2 timeleft:4 tix:30)
Random 949396 -> Winning ticket 53 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:5 tix:94) (job:1 timeleft:7 tix:73) (job:2 timeleft:3 tix:30)
Random 544177 -> Winning ticket 63 (of 197) -> Run 0

CS5600 Homework 3
Zongwang Wang

```
  Jobs:
 (* job:0 timeleft:4 tix:94 )  (  job:1 timeleft:7 tix:73 )  (  job:2 timeleft:3 tix:30 )
Random 444854 -> Winning ticket 28 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:3 tix:94 )  (  job:1 timeleft:7 tix:73 )  (  job:2 timeleft:3 tix:30 )
Random 268241 -> Winning ticket 124 (of 197) -> Run 1
  Jobs:
 (  job:0 timeleft:2 tix:94 )  (* job:1 timeleft:7 tix:73 )  (  job:2 timeleft:3 tix:30 )
Random 35924 -> Winning ticket 70 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:2 tix:94 )  (  job:1 timeleft:6 tix:73 )  (  job:2 timeleft:3 tix:30 )
Random 27444 -> Winning ticket 61 (of 197) -> Run 0
  Jobs:
 (* job:0 timeleft:1 tix:94 )  (  job:1 timeleft:6 tix:73 )  (  job:2 timeleft:3 tix:30 )
--> JOB 0 DONE at time 14
Random 464894 -> Winning ticket 55 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:6 tix:73 )  (  job:2 timeleft:3 tix:30 )
Random 318465 -> Winning ticket 92 (of 103) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:5 tix:73 )  (* job:2 timeleft:3 tix:30 )
Random 380015 -> Winning ticket 48 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:5 tix:73 )  (  job:2 timeleft:2 tix:30 )
Random 891790 -> Winning ticket 16 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:4 tix:73 )  (  job:2 timeleft:2 tix:30 )
Random 525753 -> Winning ticket 41 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:3 tix:73 )  (  job:2 timeleft:2 tix:30 )
Random 560510 -> Winning ticket 87 (of 103) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:2 tix:73 )  (* job:2 timeleft:2 tix:30 )
Random 236123 -> Winning ticket 47 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:2 tix:73 )  (  job:2 timeleft:1 tix:30 )
Random 23858 -> Winning ticket 65 (of 103) -> Run 1
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (* job:1 timeleft:1 tix:73 )  (  job:2 timeleft:1 tix:30 )
--> JOB 1 DONE at time 22
Random 325143 -> Winning ticket 3 (of 30) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:1 tix:30 )
--> JOB 2 DONE at time 23
```

./lottery.py -j 3 -s 3

```
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 3
Here is the job list, with the run time of each job:
  Job 0 ( length = 2, tickets = 54 )
  Job 1 ( length = 3, tickets = 60 )
  Job 2 ( length = 6, tickets = 6 )
** Solutions **
Random 13168 -> Winning ticket 88 (of 120) -> Run 1
  Jobs:
```

CS5600 Homework 3
Zongwang Wang

```
 (  job:0 timeleft:2 tix:54 )  (* job:1 timeleft:3 tix:60 )  (  job:2 timeleft:6 tix:6 )
Random 837469 -> Winning ticket 109 (of 120) -> Run 1
  Jobs:
 (  job:0 timeleft:2 tix:54 )  (* job:1 timeleft:2 tix:60 )  (  job:2 timeleft:6 tix:6 )
Random 259354 -> Winning ticket 34 (of 120) -> Run 0
  Jobs:
 (* job:0 timeleft:2 tix:54 )  (  job:1 timeleft:1 tix:60 )  (  job:2 timeleft:6 tix:6 )
Random 234331 -> Winning ticket 91 (of 120) -> Run 1
  Jobs:
 (  job:0 timeleft:1 tix:54 )  (* job:1 timeleft:1 tix:60 )  (  job:2 timeleft:6 tix:6 )
--> JOB 1 DONE at time 4
Random 995645 -> Winning ticket 5 (of 60) -> Run 0
  Jobs:
 (* job:0 timeleft:1 tix:54 )  (  job:1 timeleft:0 tix:--- )  (  job:2 timeleft:6 tix:6 )
--> JOB 0 DONE at time 5
Random 470263 -> Winning ticket 1 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:6 tix:6 )
Random 836462 -> Winning ticket 2 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:5 tix:6 )
Random 476353 -> Winning ticket 1 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:4 tix:6 )
Random 639068 -> Winning ticket 2 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:3 tix:6 )
Random 150616 -> Winning ticket 4 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:2 tix:6 )
Random 634861 -> Winning ticket 1 (of 6) -> Run 2
  Jobs:
 (  job:0 timeleft:0 tix:--- )  (  job:1 timeleft:0 tix:--- )  (* job:2 timeleft:1 tix:6 )
--> JOB 2 DONE at time 11
```

2. Now run with two specific jobs: each of length 10, but one (job 0) with 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?

./lottery.py -l 10:1,10:100 -c
Job 0 will rarely run before Job 1 completes. On average, Job 1 will win approximately 99 out of every 100 lotteries. In general, the job with a large amount of ticket needs will likely finish first, then the other jobs can subsequently run.

```
ARG jlist 10:1,10:100
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 0
Here is the job list, with the run time of each job:
  Job 0 ( length = 10, tickets = 1 )
  Job 1 ( length = 10, tickets = 100 )
** Solutions **
Random 844422 -> Winning ticket 62 (of 101) -> Run 1
```

CS5600 Homework 3
Zongwang Wang

```
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:10 tix:100 )
Random 757955 -> Winning ticket 51 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:9 tix:100 )
Random 420572 -> Winning ticket 8 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:8 tix:100 )
Random 258917 -> Winning ticket 54 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:7 tix:100 )
Random 511275 -> Winning ticket 13 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:6 tix:100 )
Random 404934 -> Winning ticket 25 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:5 tix:100 )
Random 783799 -> Winning ticket 39 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:4 tix:100 )
Random 303313 -> Winning ticket 10 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:3 tix:100 )
Random 476597 -> Winning ticket 79 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:2 tix:100 )
Random 583382 -> Winning ticket 6 (of 101) -> Run 1
  Jobs:
 (  job:0 timeleft:10 tix:1 )  (* job:1 timeleft:1 tix:100 )
--> JOB 1 DONE at time 10
Random 908113 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:10 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 504687 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:9 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 281838 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:8 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 755804 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:7 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 618369 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:6 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 250506 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:5 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 909747 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:4 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 982786 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:3 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 810218 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:2 tix:1 )  (  job:1 timeleft:0 tix:--- )
Random 902166 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:
 (* job:0 timeleft:1 tix:1 )  (  job:1 timeleft:0 tix:--- )
--> JOB 0 DONE at time 20
```

CS5600 Homework 3
Zongwang Wang

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.

In theory, the long-term fairness is quite high, as each job has a 50% probability of being run. However, as I observed from the results, the short term fairness is not guaranteed, as sometimes one job can simply get luck and win the lottery consistently over a short time period.

Normally, we see the finish times are close to each other, for example:
lottery.py -l 100:100,100:100 -s 1 -c

JOB 1 DONE at time 196
JOB 0 DONE at time 200

But an example with seed set to 5 can result 1 finish 19 time units faster, which is less fair.
lottery.py -l 100:100,100:100 -s 5 -c

--> JOB 1 DONE at time 181
--> JOB 0 DONE at time 200

Overall, the lottery scheduler with equal tickets essentially creates a random walk where one job will randomly get ahead and finish while the other waits. This demonstrates that while lottery scheduling is probabilistically fair, it doesn't guarantee fairness in any individual execution - a fundamental tradeoff of randomized algorithms.

4. How does your answer to the previous question change as the quantum size (-q) gets larger?

As the quantum size increases, the unfairness in lottery scheduling with equal tickets actually decreases significantly. Longer quantum size means less frequency of the lottery. Imagine having 100 lottery vs having only 1 lottery, the overall fairness level will be reduced, as the completion time will vary by a lot.

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?

CS5600 Homework 3
Zongwang Wang