CS5600 MLFQ Report - Team 1

This group report includes the following parts:

Description:
- Policies
- Test cases

Evaluation:
- Completion time
- Response time

Group member: Zhuojin Liu

Policies

My MLFQ is similar to the OSTEP MLFQ in most aspects, but there are two key differences I would like to highlight.

The first difference is my MLFQ added a new flag, -g, which means gaming tolerance. By adding this flag, we can tolerate jobs gaming with the MLFQ to some extent to keep interactive jobs on the high-level queue. This flag means if a job issues an I/O before reaching gamingTolerance% of quantum length, its allotment won't be reduced. This is a moderate solution to the OSTEP simulator's -S flag, which can only completely tolerate or intolerate gaming. The benefit of this flag can be observed from the first test case below.

The second difference is that my MLFQ could customize I/O time for each job instead of a fixed I/O time for all jobs. In this way, we can test more complicated situations where some jobs issue frequent but short I/O (interactive programs such as UI) and others issue less frequent but more arduous I/O (e.g., saving a trained model to disk). This is demonstrated in the second test case.

Test cases

test case 1:

My MLFQ:

`./main -j 0,1000,0:50,100,1,10:30,1000,0 -g 10 -b 100000`

```
Job 1 turnaround time: 2090, response time: 0
Job 2 turnaround time: 1090, response time: 0
Job 3 turnaround time: 2070, response time: 0
```

OSTEP MLFQ:

`python3 mlfq.py -l 0,1000,0:50,100,1:30,1000,0 -i 10 -c -B 100000`

```
Job  0: startTime   0 - response   0 - turnaround 2057
Job  1: startTime  50 - response   0 - turnaround 2260
Job  2: startTime  30 - response   0 - turnaround 2048
```

test case 2:

`./main -j 0,1000,0:50,100,1,10:30,1000,0:200,1000,500,100 -b 100000`

```
Job 1 turnaround time: 2930, response time: 0
Job 2 turnaround time: 1096, response time: 0
Job 3 turnaround time: 2890, response time: 0
Job 4 turnaround time: 2900, response time: 0
```

Group member: Yufei Zhang
Policies: Implemented all the aspect of OSTEP MLFQ except the IO part. Enabled customizable job length, number of queues, allotment for each queue level, quantum for each queue level. Implemented boost time but does not have big effect on scheduling without the gameplay of IO.

Test cases

1.

-n 3 -q 10 -a 5 -j 0,50:0,100:0,100 -b 0

Job 0: startTime 0 - response 0 - turnaround 148
Job 1: startTime 0 - response 1 - turnaround 249
Job 2: startTime 0 - response 2 - turnaround 250

Avg: startTime n/a - response 1 - turnaround 215.667

2.

-n 5 -q 10 -a 2,3,4,5,6 -j 0,100:0,300:0,100:0,200:0,400

Job 0: startTime 0 - response 0 - turnaround 496
Job 1: startTime 0 - response 1 - turnaround 999
Job 2: startTime 0 - response 2 - turnaround 498
Job 3: startTime 0 - response 3 - turnaround 799
Job 4: startTime 0 - response 4 - turnaround 1100

Avg: startTime n/a - response 2 - turnaround 778.4

Group member: Lu Yan

Policies
This simulator employs a round-robin scheduling policy with multilevel feedback queues.
4 of 5 rules in text book are implemented:
Rule 1: If Priority(A) > Priority(B) run A.
Rule 2: If Priority(A) == Priority(B), run them in Round-Robin.
Rule 3: When a job enters the system, it is placed at the highest priority.
Rule 5: After some time period S, move all of the jobs in the system to the topmost queue.
This scheduler has 5 queues, each with allotment, quantum as follows:
Q0: 10,5
Q1: 24,8
Q2: 40,10
Q3: 75,15
Q4: N/A, 20
Voo-Doo (boost) time is 40

Note: this simulator have 3 drawbacks and will be improved soon
(1) cannot specify job start time, which means it can only assume all processes enter the same time
(2) cannot simulate io
(3) the bottom queue still has the method demote(), so when creating the bottom queue, its allotment must be specified a very big number.

How to play with cpu scheduler simulator:
You can adjust the parameters of the queues and running processes by reading the following instructions.

How to tune queues:
From line 135-144, queues are created with allotment, quantum, level, and its container mlfq, you can use your own values to tune the queues.
Syntax: Queue(allotment, quantum, level, mlfq)
Here you can create as many queues you want, but the top queue should be called q0 have have level 0.
e.g Queue* q0 = new Queue(your number,your number, 0, mlfq);
And the other queues have levels 1,2,3,4....
Note the bottom queue must have a very long allotment:
e.g. Queue* q4 = new Queue(10000000,your number, your number,mlfq);

How to add processes:
From line 146-159, processes are created with pid and length:
Syntax : Process(pid,p_length)
Here you can add, reduce the number of processes, and specify your own pid and length of process.

Adjust VOO-DOO (boost) time:
You can modify the VOO_DOO constant at line 8.

After specifying all your parameters above, you can just compile and run. Every process and its final completion time, response time will be printed.

Test cases

Lengths of processes: 5, 15, 30, 60, 100, 150 (all start at time 0)

Completion time summary:
1001 is completed at 5
1002 is completed at 60
1003 is completed at 116
1004 is completed at 222
1005 is completed at 302
1006 is completed at 360
Average completion time: 177.5
Response time summary:
1001 is respond at 0
1002 is responded at 5
1003 is responded at 10
1004 is responded at 15
1005 is responded at 20
1006 is responded at 25
Average response time: 12.5

Group member: Liting Zhou

Description:
Policies
My MLFQ implements the following rules:
Rule 1: If Priority(A) > Priority(B), A runs (B doesn't).
Rule 2: If Priority(A) = Priority(B), A & B run in round-robin fashion using the time slice (quantum length) of the given queue.

Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue).
Rule 4: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
Rule 5: After some time period S, move all the jobs in the system to the topmost queue.

The differences from the OSTEP mlfq.py include:
1. I only check for available jobs with higher priorities after a quantum finished once the current job starts running. This might harm the performance of average response time, but improve the turnaround time.
2. Similar the the 1st point, my boost does not execute before a job finishes its started quantum.

Test cases
-l 0,20,0:3,3,0:5,8,2 -Q 1,2,4 -i 2 -B 5 -c
Evaluation:
My MLFQ: Avg response 0.67 - turnaround 18.67
OSTEP: Avg response 0.67 - turnaround 19.00

-l 0,20,0:3,3,0:5,8,2 -Q 1,2,4 -i 2 -B 10 -c
Evaluation:
My MLFQ: Avg response 0.33 - turnaround 18.33
OSTEP: Avg response 0.00 - turnaround 18.67