CSS 430 Operating Systems

Program 2 Report

Donghee Lee

Scheduler.java

Scheduler.java is a program for ThreadOS that behaves similarly to a scheduler for Operating Systems. In this project, I implement the Scheduler using the Round-Robin and Multilevel Feedback Queue algorithm.

Part 1: Implementing a Round-Robin (RR) Scheduler

1. Algorithm

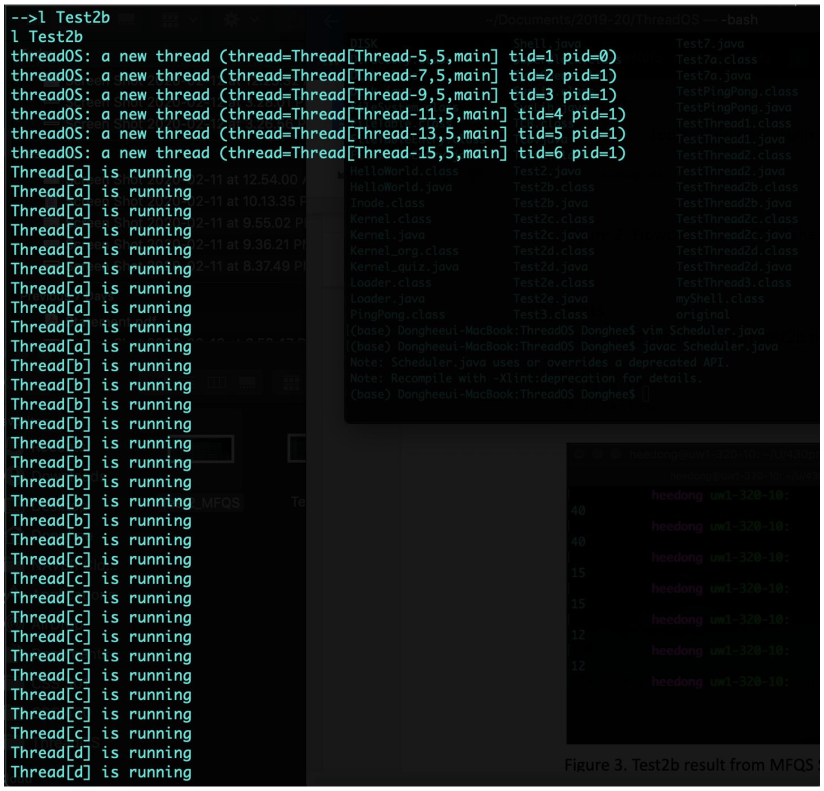
With the Round-Robin algorithm, the scheduler schedules processes or threads by allowing them the same amount of time quantum. If a job is not completed during the given time quantum, it will be put in the back of the queue to be executed later. Since all threads are allocated the same amount of time in turn, this algorithm is fair and avoids starvation.

2. Test Result

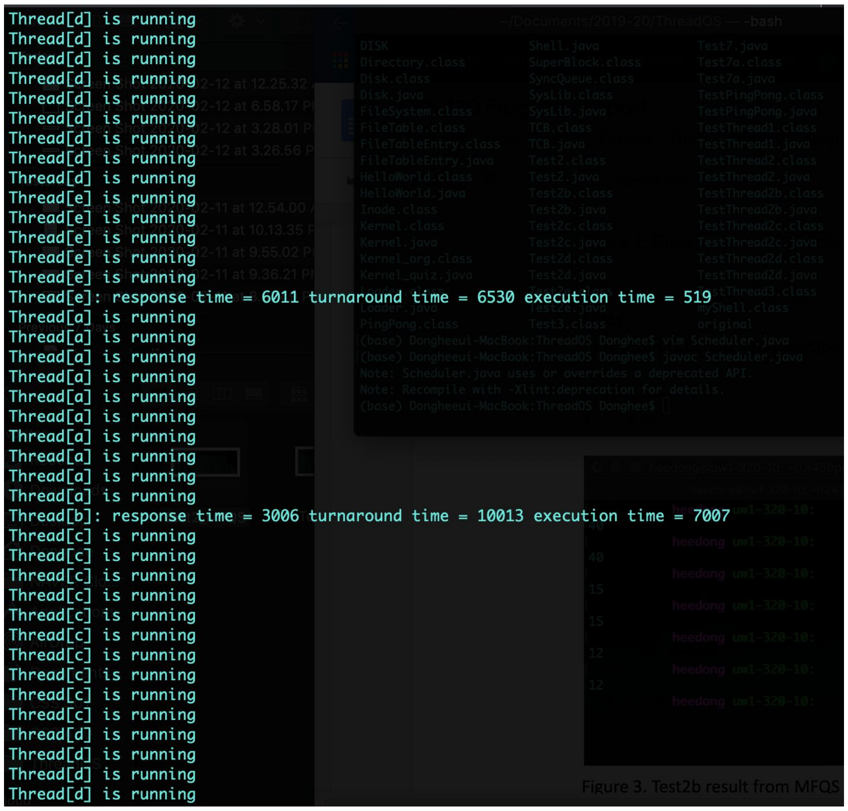
To test this program, I observe the result of running Test2b on ThreadOS:

```
$ java Boot
$ l Test2b
```

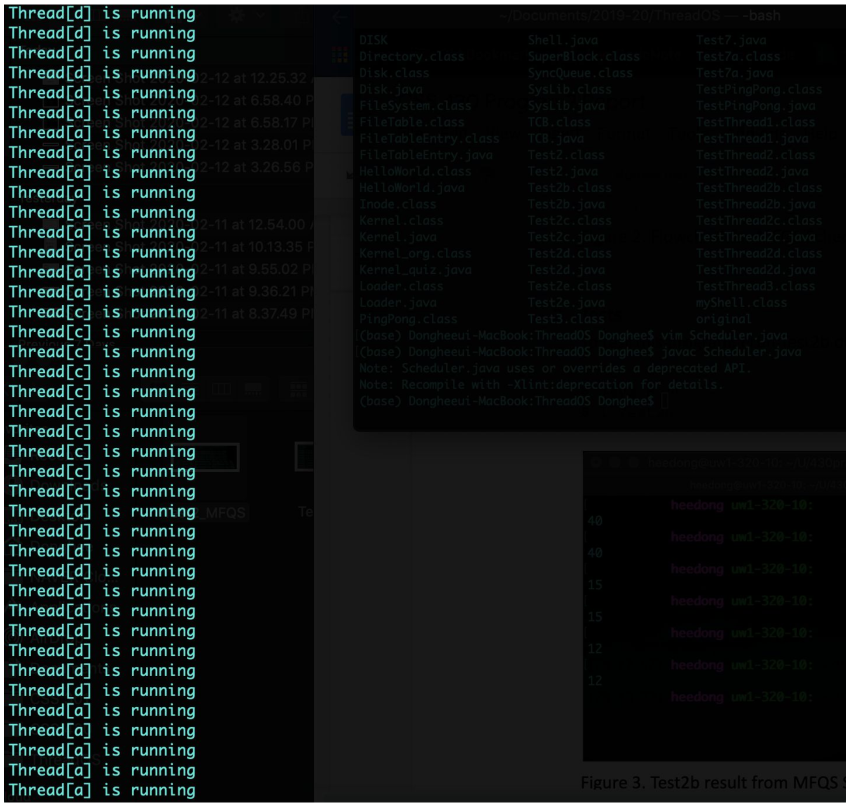
From Figure 1, it is shown that the threads are given the same amount of quantum, and the context is switched if they do not finish in the allowed time. If they finish before using up the given quantum, the next thread is run.



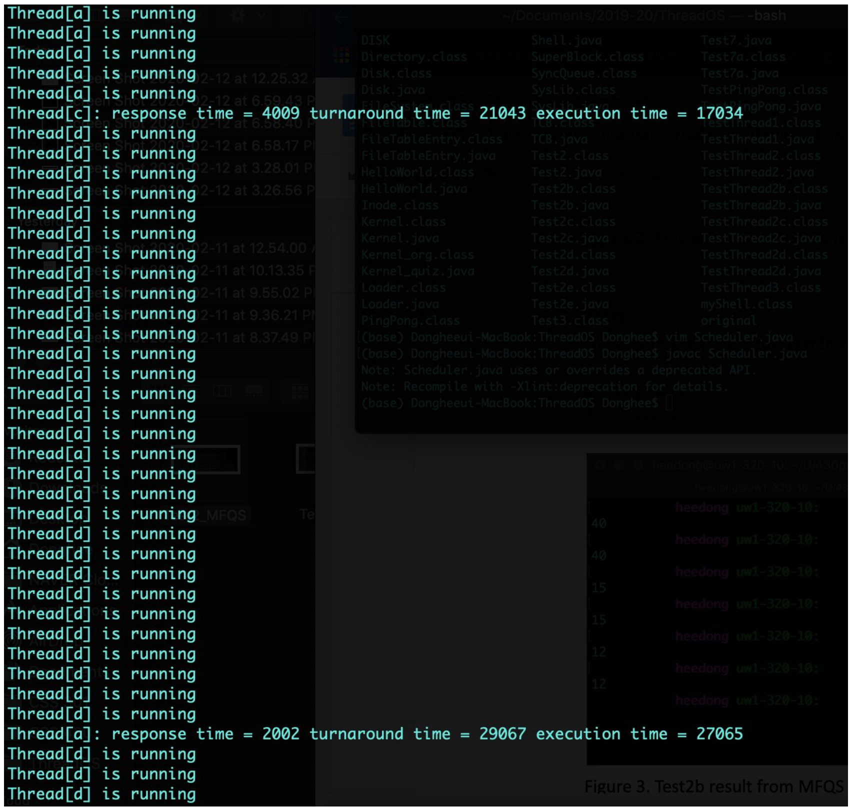
```
-->l Test2b
l Test2b
threadOS: a new thread (thread=Thread[Thread-5,5,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,5,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,5,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,5,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,5,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,5,main] tid=6 pid=1)
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[d] is running
Thread[d] is running
```



```
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[e]: response time = 6011 turnaround time = 6530 execution time = 519
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[b]: response time = 3006 turnaround time = 10013 execution time = 7007
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
```



```
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
```



```
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[c]: response time = 4009 turnaround time = 21043 execution time = 17034
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[a]: response time = 2002 turnaround time = 29067 execution time = 27065
Thread[d] is running
Thread[d] is running
Thread[d] is running
```

Figure 1. Test2b Result from RR Scheduler

Part 2: Implementing a Multilevel Feedback Queue (MFQS) Scheduler

1. Algorithm

With the Multilevel Feedback Queue algorithm, the scheduler schedules processes or threads by their priorities. In this project, I have three priorities: 0, 1, and 2. Queue 0 contains threads with higher priorities while Queue 2 contains threads with lower priorities. All new threads will first be assigned to Queue 0. Threads in Queue 1 will be run only if Queue 0 is empty. Similarly, threads in Queue1 will be scheduled only if Queue 2 is empty.

The time quantum for the threads is 500 milliseconds in Queue 0, 1000 milliseconds in Queue 1, and 2000 milliseconds in Queue 2. If the first thread of Queue 0 is not finished in the given quantum, it is put at the end of the Queue 1. If the first thread of Queue 1 is not finished in the given quantum, it is put at the end of the Queue 2. If a thread in the front of the Queue2 is not finished in the given quantum, it is put at the end of the Queue 2 again.

This MFQS scheduler allows preemption. That is, the scheduler checks every 500 milliseconds if there is a new thread in the higher queue(s) while it is running the thread in Queue 1 or Queue 2. While a thread in the lower queue is running, if the scheduler finds that there is a thread ready in the higher queue(s), the current thread will be suspended until the thread(s) in the higher queue(s) are run. When all of the threads in the higher queue(s) are finished, the suspended thread will be resumed and given the rest of the time quantum it had left.

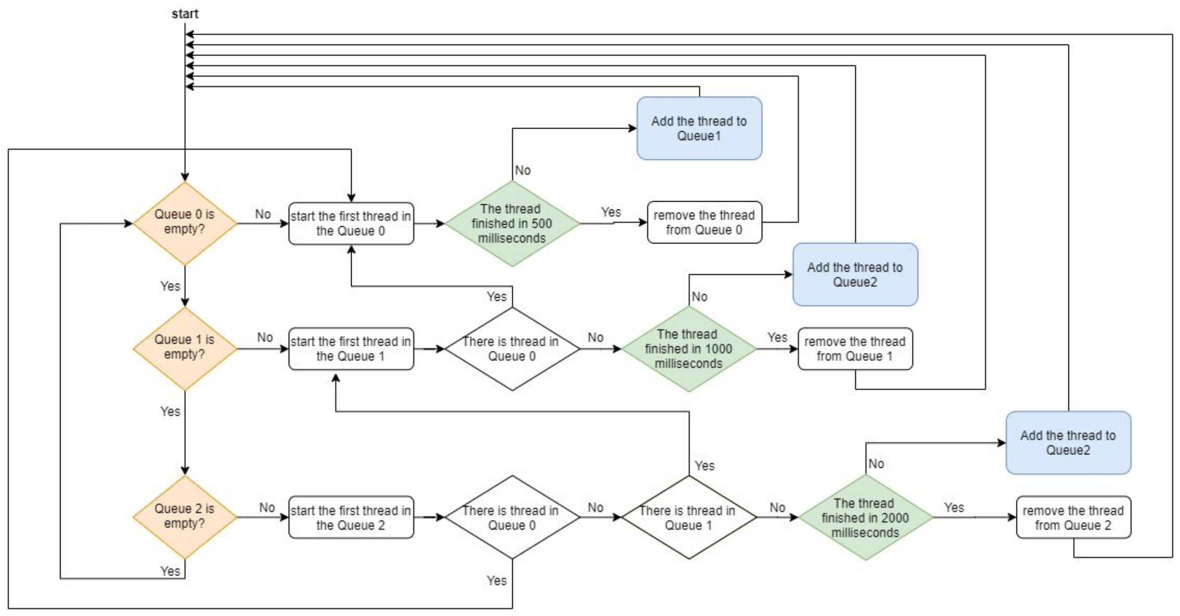

start
Queue 0 is empty?
No
start the first thread in the Queue 0
The thread finished in 500 milliseconds
No
Add the thread to Queue1
Yes
remove the thread from Queue 0
Yes
Queue 1 is empty?
No
start the first thread in the Queue 1
There is thread in Queue 0
Yes
No
The thread finished in 1000 milliseconds
No
Add the thread to Queue2
Yes
remove the thread from Queue 1
Yes
Queue 2 is empty?
No
start the first thread in the Queue 2
There is thread in Queue 0
Yes
No
There is thread in Queue 1
Yes
No
The thread finished in 2000 milliseconds
No
Add the thread to Queue2
Yes
remove the thread from Queue 2
Yes

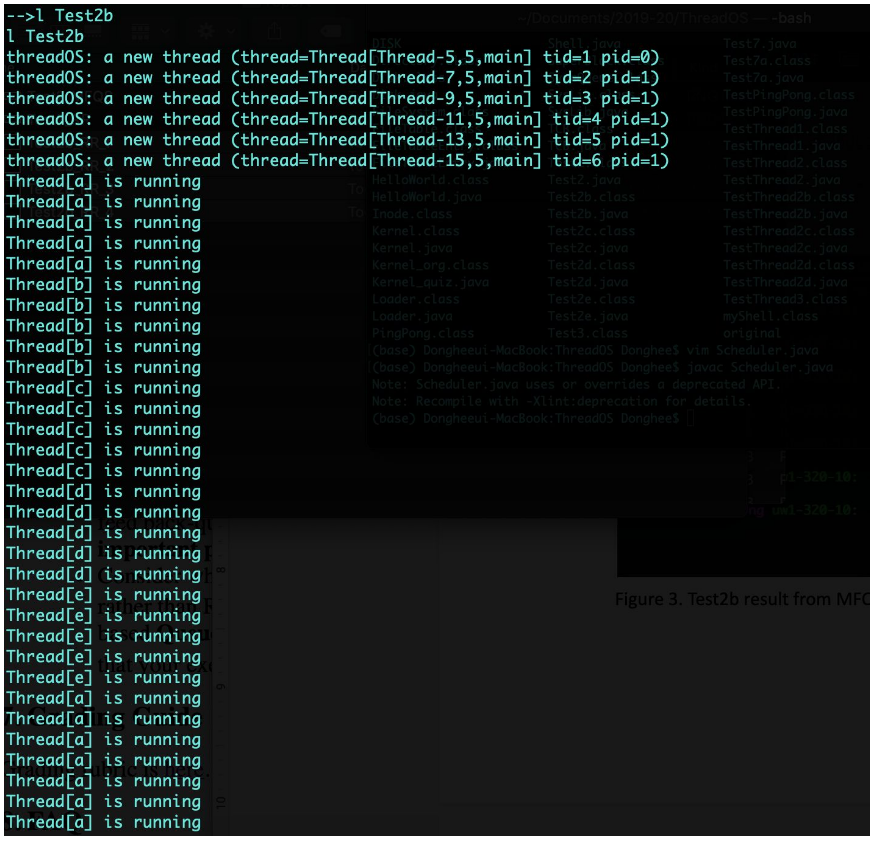
Figure 2. Flowchart for MFQS Scheduler

2. Tests

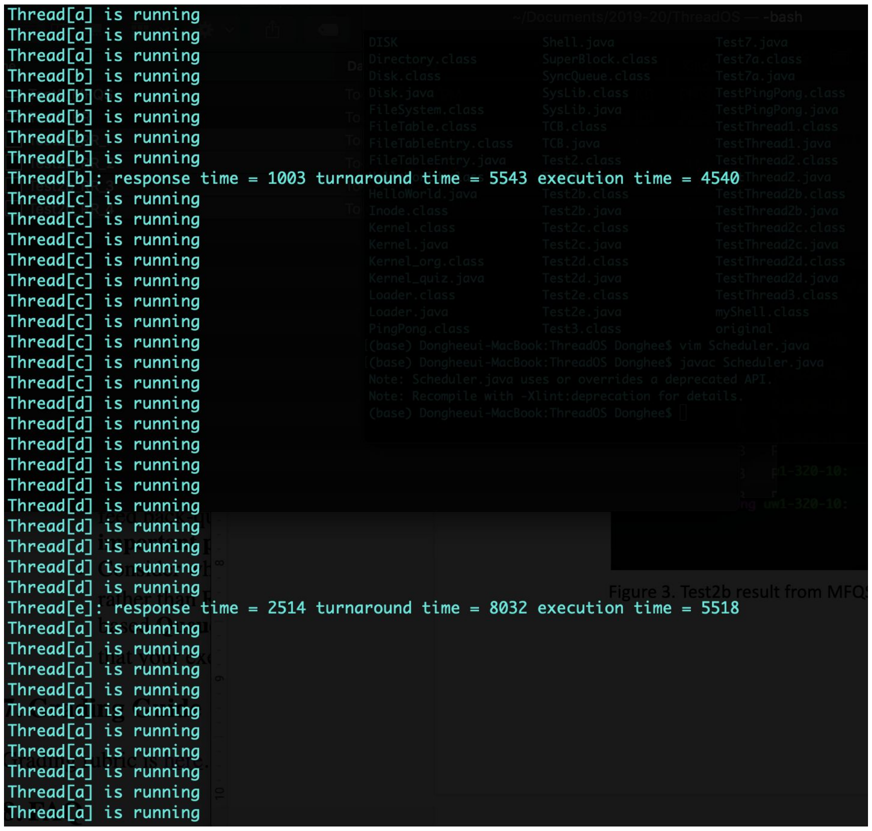
To test this program, I observe the result of running Test2b on ThreadOS:

```
$ java Boot
$ l Test2b
```

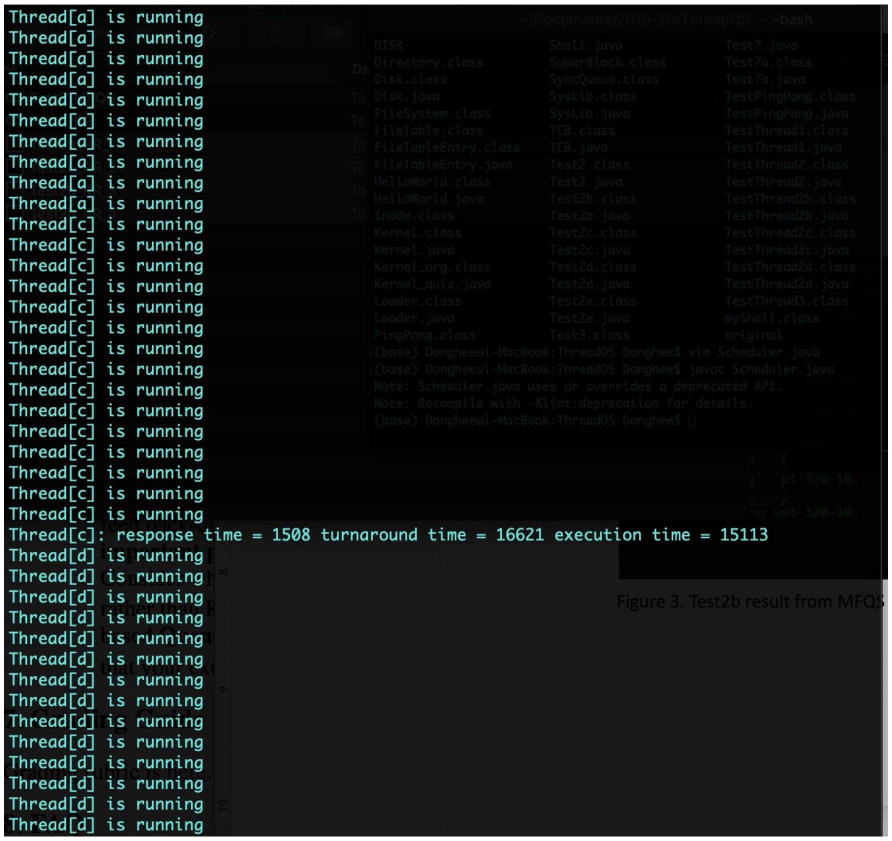
From Figure 3, it is shown that threads a, b, c, d, and e are put into Queue 0 and thereafter given 500ms. If the threads aren't finished in the given time, they're put into Queue 1 and thereafter given 1000ms. Finally, if the threads aren't finished in the given time, they're put into Queue 2 and given 2000ms. The threads will stay in Queue 2 until they finish their jobs.



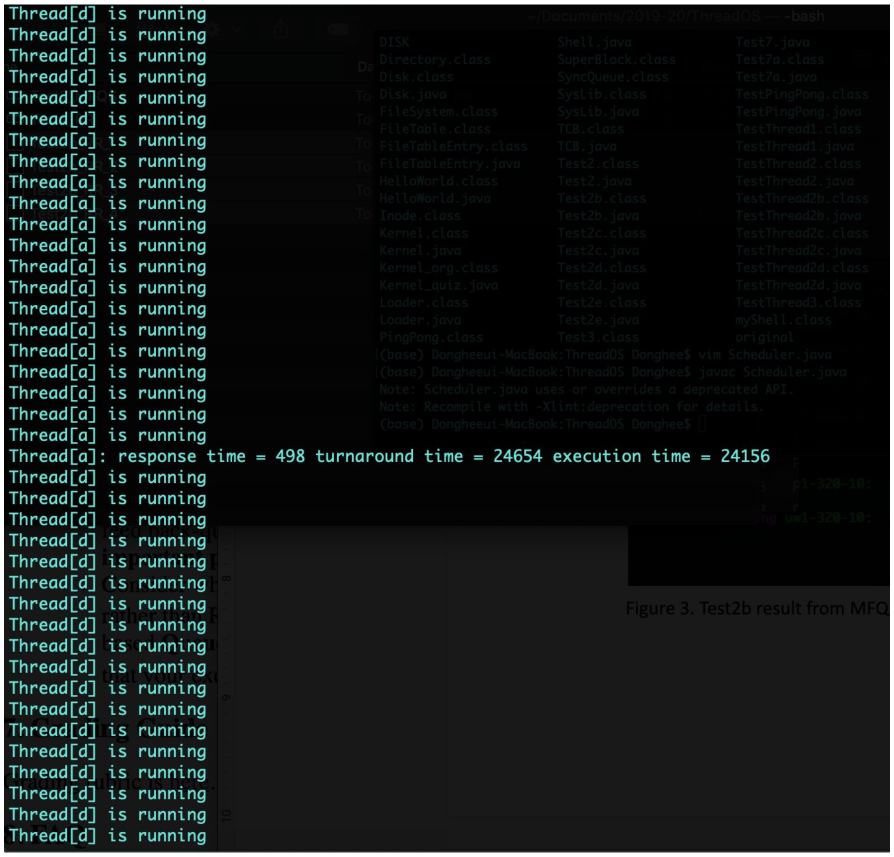
```
-->l Test2b
l Test2b
threadOS: a new thread (thread=Thread[Thread-5,5,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,5,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,5,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,5,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,5,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,5,main] tid=6 pid=1)
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[e] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
```



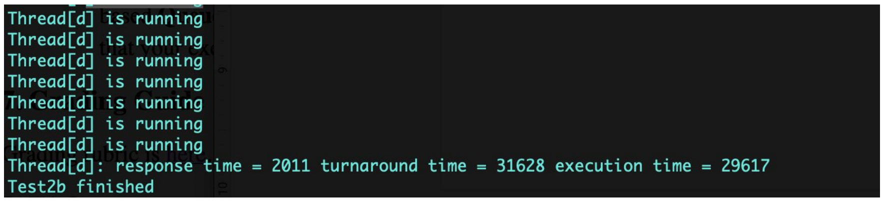
```
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b] is running
Thread[b]: response time = 1003 turnaround time = 5543 execution time = 4540
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[e]: response time = 2514 turnaround time = 8032 execution time = 5518
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
```



```
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c] is running
Thread[c]: response time = 1508 turnaround time = 16621 execution time = 15113
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
```



```
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a] is running
Thread[a]: response time = 498 turnaround time = 24654 execution time = 24156
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
```



```
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d] is running
Thread[d]: response time = 2011 turnaround time = 31628 execution time = 29617
Test2b finished
```

Figure 3. Test2b result from MFQS Scheduler

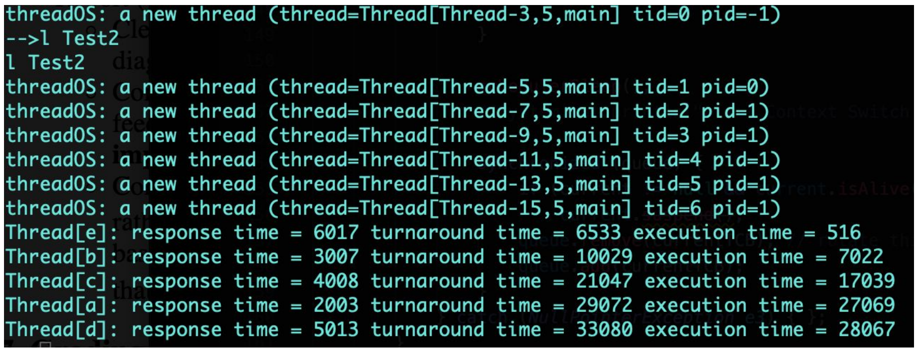
Part 3: Comparison between MFQS and Round-Robin Schedulers

1. Test Results

Based on Figure 4 and Figure 5, it is clear that the MFQS scheduler has a better (less) response and turnaround time compared to the RR scheduler. However, the RR scheduler has a better (less) execution time than the MFQS scheduler.

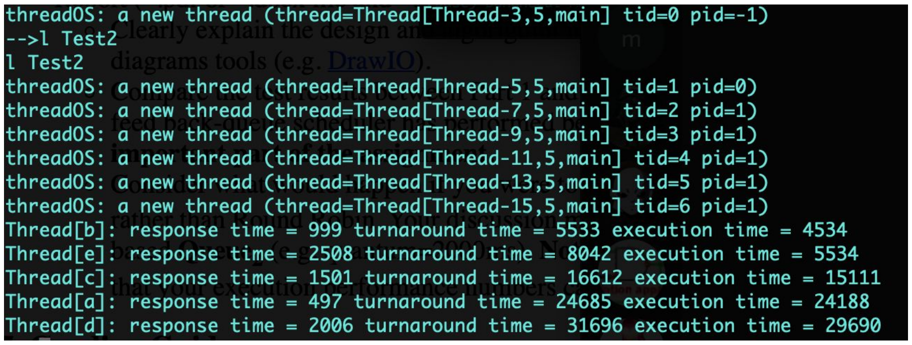
The MFQS scheduler has a shorter response and turnaround time because it prioritizes the threads that take a shorter amount of time. When such threads are prioritized, it reduces the average waiting time and reduces response time. Also, from Figure 6, we can see that there are less context switches in MFQS than RR, making it have less turnaround time.

The RR scheduler having a shorter execution time in this test can be explained with this equation: *execution time = turnaround time - response time*. The RR scheduler has less execution time because while turnaround time for both are similar in this test, the response time in RR is way more than one in MFQS.



```
threadOS: a new thread (thread=Thread[Thread-3,5,main] tid=0 pid=-1)
-->l Test2
l Test2
threadOS: a new thread (thread=Thread[Thread-5,5,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,5,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,5,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,5,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,5,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,5,main] tid=6 pid=1)
Thread[e]: response time = 6017 turnaround time = 6533 execution time = 516
Thread[b]: response time = 3007 turnaround time = 10029 execution time = 7022
Thread[c]: response time = 4008 turnaround time = 21047 execution time = 17039
Thread[a]: response time = 2003 turnaround time = 29072 execution time = 27069
Thread[d]: response time = 5013 turnaround time = 33080 execution time = 28067
```

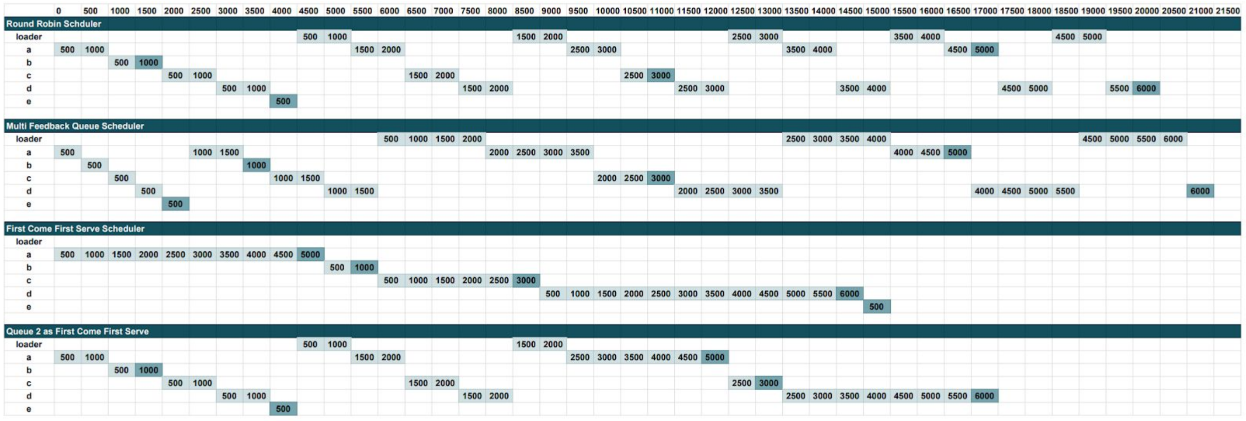
Figure 4. Test2 result from RR Scheduler



```
threadOS: a new thread (thread=Thread[Thread-3,5,main] tid=0 pid=-1)
-->l Test2
l Test2
threadOS: a new thread (thread=Thread[Thread-5,5,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,5,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,5,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,5,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,5,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,5,main] tid=6 pid=1)
Thread[b]: response time = 999 turnaround time = 5533 execution time = 4534
Thread[e]: response time = 2508 turnaround time = 8042 execution time = 5534
Thread[c]: response time = 1501 turnaround time = 16612 execution time = 15111
Thread[a]: response time = 497 turnaround time = 24685 execution time = 24188
Thread[d]: response time = 2006 turnaround time = 31696 execution time = 29690
```

Figure 5. Test2 result from MFQS Scheduler

Figure 6 shows a Gantt chart for Round-Robin scheduler (quantum = 1000 ms), MFQ scheduler (quantum 0 = 500 ms, quantum 1 = 1000 ms, quantum 2 = 2000 ms), FCFS scheduler, and MFQ scheduler with Queue 2 as FCFS.



	0	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000	10500
Round Robin Schduler																						
loader										500	1000							1500	2000			
a	500	1000										1500	2000							2500	3000	
b			500	1000																		
c					500	1000								1500	2000							2500
d							500	1000								1500	2000					
e									500													
Multi Feedback Queue Scheduler																						
loader													500	1000	1500	2000						
a	500					1000	1500											2000	2500	3000	3500	
b		500						1000														
c			500						1000	1500											2000	2500
d				500							1000	1500										
e					500																	
First Come First Serve Scheduler																						
loader																						
a	500	1000	1500	2000	2500	3000	3500	4000	4500	5000												
b											500	1000										
c													500	1000	1500	2000	2500	3000				
d																			500	1000	1500	2000
e																						
Queue 2 as First Come First Serve																						
loader										500	1000							1500	2000			
a	500	1000										1500	2000							2500	3000	3500
b			500	1000																		
c					500	1000								1500	2000							
d							500	1000								1500	2000					
e									500													

	11000	11500	12000	12500	13000	13500	14000	14500	15000	15500	16000	16500	17000	17500	18000	18500	19000	19500	20000	20500	21000	21500
Round Robin Schduler																						
loader				2500	3000					3500	4000					4500	5000					
a						3500	4000					4500	5000									
b																						
c	3000																					
d		2500	3000					3500	4000					4500	5000			5500	6000			
e																						
Multi Feedback Queue Scheduler																						
loader						2500	3000	3500	4000								4500	5000	5500	6000		
a										4000	4500	5000										
b																						
c	3000																					
d		2000	2500	3000	3500								4000	4500	5000	5500					6000	
e																						
First Come First Serve Scheduler																						
loader																						
a																						
b																						
c																						
d	2500	3000	3500	4000	4500	5000	5500	6000														
e									500													
Queue 2 as First Come First Serve																						
loader																						
a	4000	4500	5000																			
b																						
c				2500	3000																	
d						2500	3000	3500	4000	4500	5000	5500	6000									
e																						

Figure 6. Gantt Chart for RR, MFQS, and FCFS Schedulers

2. Considering the Last Queue as a First-Come First-Serve (FCFS)

If the last queue (Queue 2) were implemented as First-Come First-Serve, threads that arrive to the CPU earlier will occupy the CPU until they finish. If those only require a short time, the average wait time, turnaround time, or response time would not suffer. However, they will suffer if the earlier threads require a long time since Queue 2 is not preemptive. Even if new threads are added to Queue 0, the scheduler will not suspend the current thread in Queue 2, increasing the average wait time.