Name: Thuan Tran
Date: April 23th, 2017
CSS 430 Program 2

REPORT

Table of Contents

Purpose of the program

Implement a Scheduler for the ThreadOS. A scheduler is something that will schedule the process to run and preempt them for another thread/process to run
This assignment consists of three parts total

Part 1

For this part, the assignment only requires to change several lines in the run method so that the Scheduler will act like a round robin rule. Previous implementation did not strictly enforce round-robin rule which resulted in other processes interleaved with each other and produce results that were interleaved as well. After this modification, the processes is ensured to act in a round robin fashion where the processes occur sequentially and they were given a time slice. After that time slice, the process is suspended and the scheduler pick another process to run it

Below is the execution of the Scheduler after modification running Test2b

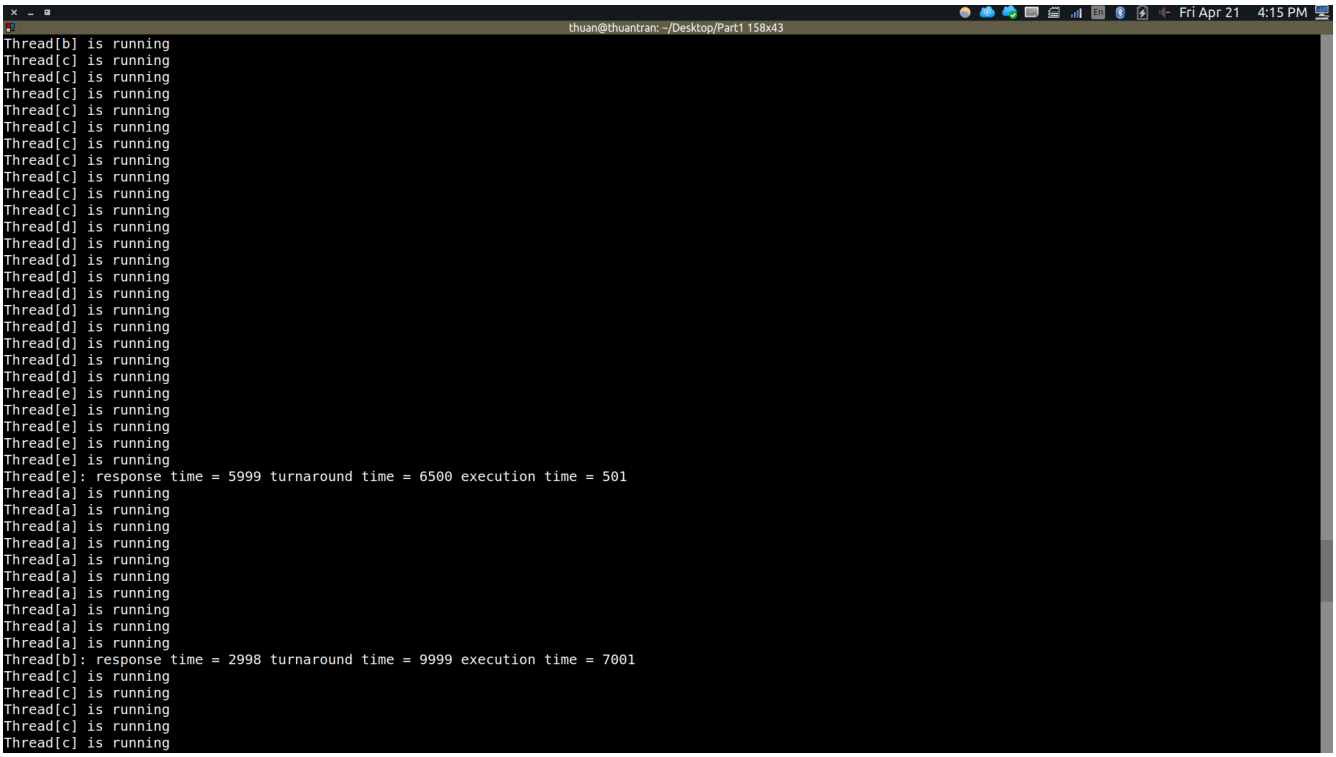

Illustration 1: Part 1 Output with Test2b

Part 2

For this part of the assignment, we need to implement the Scheduler using a multi level queue feedback. To be specific, this algorithm will require 3 queues where as the round-robin algorithm only require a single queue. For this algorithm to work, here are the places where I made changes:

- I declared 3 instance variables represent 3 queues
- I modified the run method
- I modified the getMyTCB method
- I modified the constructor

Modified constructor

There is nothing change much where I only add the initialization of the new queues

Modified getMyTCB

In the previous implementation, the getMyTCB method only work for 1 queue. Now I added 2 new for loops that search through the TCB in those 2 new queues

Modified run method

For the run method , I decided to create 3 new private method that is called executeFirstQueue(), executeSecondQueue() and executeThirdQueue() and their functionalities are what their name suggest

executeFirstQueue()

This method works like the previous implementation of the part 1 where it will first get the TCB of the thread. Check to see if the TCB is valid and get its thread. Then it will perform additional check to see if the threads are valid. If it is valid, then resume its operation otherwise start it
After that, I called the sleep method for half of the time slice. After the sleep method, I put the synchronized keyword to make sure that the first queue is synchronized. Then I check the thread is still alive. If the thread is still alive, its mean that it did not finish its execution in that time slice. Then I moved that thread to the second queue

executeSecondQueue()

This method is similar like the executeFirstQueue method above until the point where I called the sleep method for half of the time slice. After that, I check to see if the size of the first queue is not 0. If it is, then I suspend the current thread and call the first queue method to let the first queue execute all of its jobs. After that, I resumed the thread and called the sleep method for addition half of the time slice
The remaining part of the method is similar where I use the synchronized key word, check the thread, … If the thread did not finish, then move it out of second queue and add to third queue

executeThirdQueue()

This method is also similar to the above 2 methods all the way until it check if the thread is valid or not to resume or start it.
However, this method now also create a while loop that run 4 times, each time represent half of the time slice. During each time, it will call the sleep method to sleep half of the time slice. Then proceed to check if the first and second queue is not empty. If they are not empty, then suspend the thread and call the executeFirstQueue and executeSecondQueue method. Then resume the thread and end of 1 iteration,
During the synchronized check of the third queue, if the thread is not yet done, suspend it,remove it from the queue and add it back to the queue (the tail)

Output

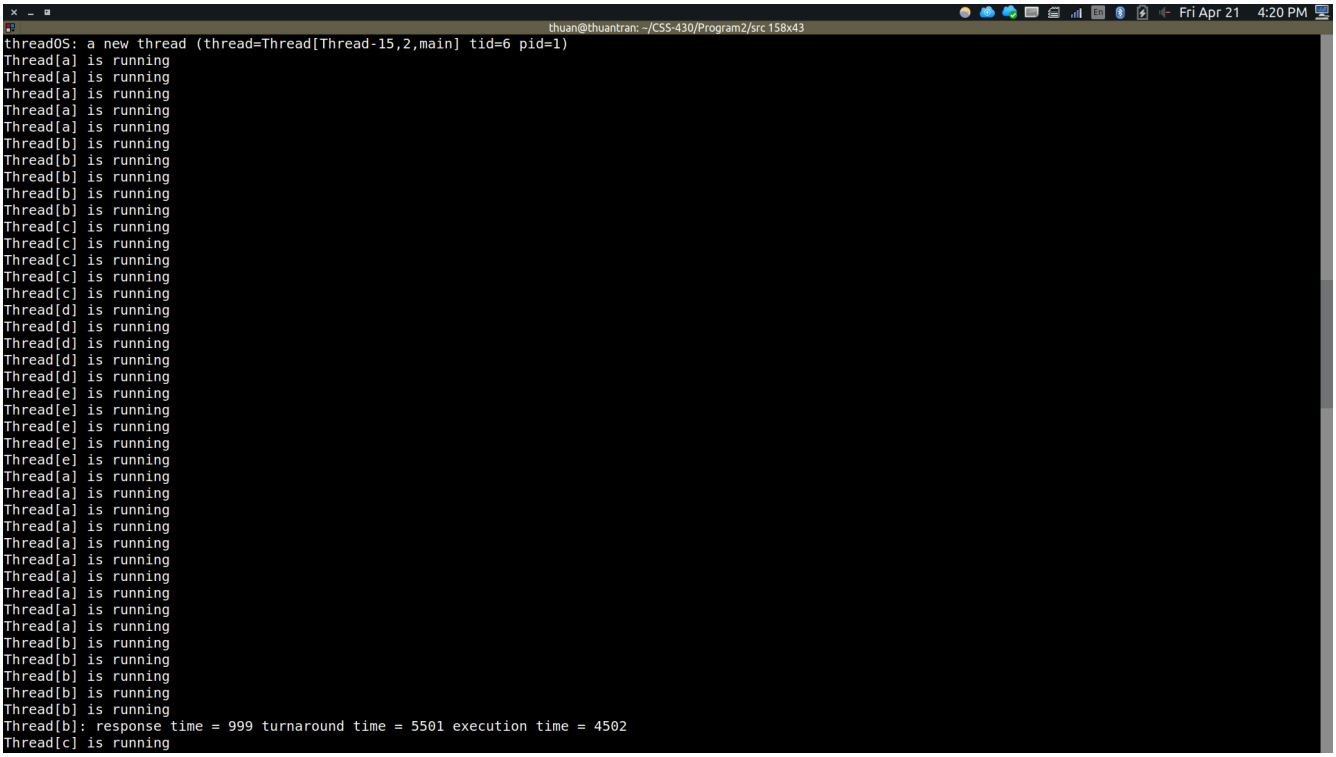

Illustration 2: MultiLevelQueue with Test2b

Part 3

For this part. I need to compare the performance of the algorithm in part 1 and part 2 using Test2
Below are the out put when I ran Test 2 using 2 different algorithms and a summary table of the output

```
thuan@thuantran: ~/Desktop/Part1 158x43
Directory.class       PingPong.java       Test2d.class  TestThread1.class
Disk.class            QueueNode.class     Test2d.java   TestThread1.java
Disk.java             Scheduler.class     Test2e.class  TestThread2b.class
FileSystem.class      Scheduler_fil.java  Test2e.java   TestThread2b.java
FileTable.class       Scheduler.java      Test2.java    TestThread2c.class
FileTableEntry.class  Shell.class         Test3.class   TestThread2c.java
FileTableEntry.java   SuperBlock.class    Test4.class   TestThread2.class
HelloWorld.class      SyncQueue.class     Test5.class   TestThread2d.class
HelloWorld.java       SysLib.class        Test5.java    TestThread2d.java
Inode.class           SysLib.java         Test6.class   TestThread2.java
Kernel.class          TCB.class           Test6.java    TestThread3.class
Kernel.java           TCB.java            Test7a.class
Kernel_org.class      Test2b.class        Test7a.java
thuan@thuantran:~/Desktop/Part1$ nano Scheduler.java
Use "fg" to return to nano.

[3]+  Stopped                 nano Scheduler.java
thuan@thuantran:~/Desktop/Part1$ javac Scheduler.java
Note: Scheduler.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: Scheduler.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
thuan@thuantran:~/Desktop/Part1$ java Boot
threadOS ver 1.0:
threadOS: DISK created
default format( 64 )
Superblock synchronized
Type ? for help
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 = 6000 turnaround time = 6501 execution time = 501
Thread[b]: response time = 2999 turnaround time = 10000 execution time = 7001
Thread[c]: response time = 4000 turnaround time = 21002 execution time = 17002
Thread[a]: response time = 1999 turnaround time = 29003 execution time = 27004
Thread[d]: response time = 5000 turnaround time = 33003 execution time = 28003
-->
```

Illustration 3: Part 1 Output with Test2

```
thuan@thuantran: ~/CSS-430/Program2/src 158x43
Thread[a] is running
Thread[a]: response time = 499 turnaround time = 23006 execution time = 22507
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 = 1999 turnaround time = 30005 execution time = 28006
Test2b finished
-->l Test2
l Test2
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=0)
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=7)
threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=7)
threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=7)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=7)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=7)
Thread[b]: response time = 1000 turnaround time = 5502 execution time = 4502
Thread[e]: response time = 2500 turnaround time = 8001 execution time = 5501
Thread[c]: response time = 1500 turnaround time = 14504 execution time = 13004
Thread[a]: response time = 500 turnaround time = 22507 execution time = 22007
Thread[d]: response time = 2000 turnaround time = 29506 execution time = 27506
-->
```

Illustration 4: Part 2 Output with Test2

	Execution	Response Time	Turn around Time	BurstTime
Thread a	22007	500	22507	5000
Thread b	4502	1000	5502	1000
Thread c	13004	1500	14504	3000
Thread d	27506	2000	29506	6000
Thread e	5501	2500	8001	500
Thread a	27004	1999	29003	5000
Thread b	7001	2999	10000	1000
Thread c	17002	4000	21002	3000
Thread d	28003	5000	33003	6000
Thread e	501	6000	6501	500

Illustration 5: Summary Table of Part 2 (First 5 row) and Part 1 (last 5 row) using Test2

Looking at the response time, the multi level queue feed back performs better than the round robin one because each processes in the multil level queue was only given half of the time slice and once it is over, another process took place of it and that process is pushed down to another queue. And in the round-robin, each process was given the whole time slice before another process can take place of it.

Looking at other stats, I can't really tell which one of those two is better since they all have their up and down. Mostly all threads using the multi-level queue has their turn around time less than the one using the round robin (except thread e , mostly because its burst time was within the time quantum)

So depending on the context, pick either the round robin or the multi-level one. The round robin is suitable if you want to process the processes in a round robin style where each thread has the same priority and they were executed based on their order of arrival. Choose the feedback queue if you want to prioritize some processes over other. Those initial arrived processes will be put onto other queues, having their time slice increase but priority decrease.

What happen if FCFS ?

If Part 2 was implemented using a FCFS order, then the response time of each elements will go up because it will wait until the other process finished before a new process begin
The execution time will reduce to the exact burst time because there will be no interrupt during its execution (from its start of response time to its termination)
However, this might cause problem if some really big processes get in first because they might use all the resources and block other process to run.