Round Robin(RR) and Multilevel Feedback Queue Scheduler (MFQS)

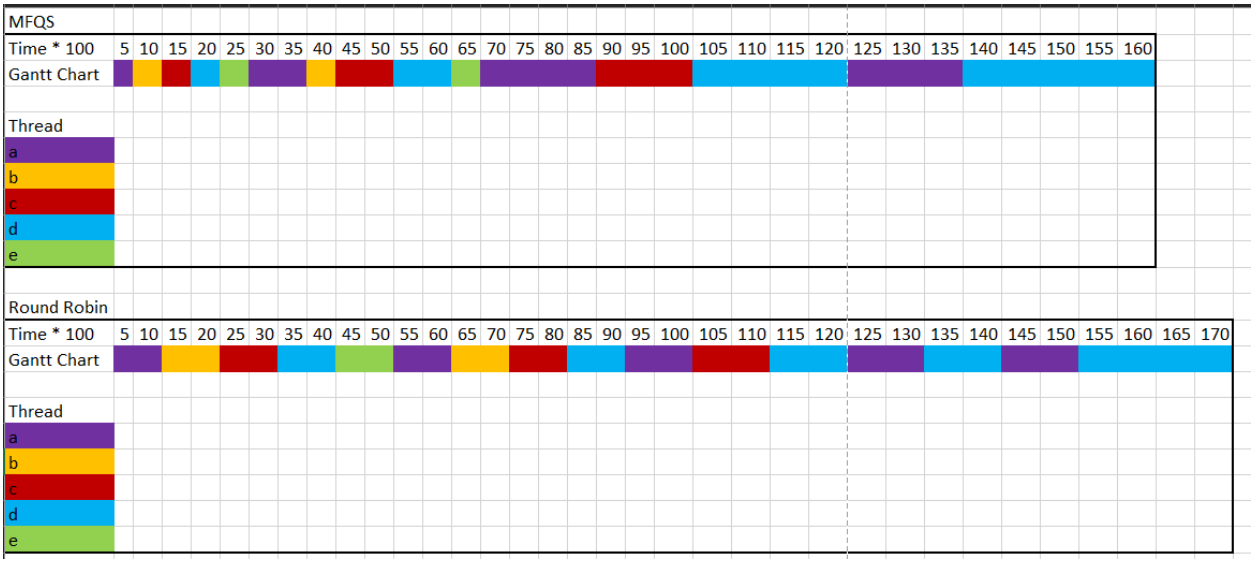
RR:

```
l Test2
threadOS: a new thread (thread=Thread[Thread-17,5,main] tid=7 pid=0)
threadOS: a new thread (thread=Thread[Thread-19,5,main] tid=8 pid=7)
threadOS: a new thread (thread=Thread[Thread-21,5,main] tid=9 pid=7)
threadOS: a new thread (thread=Thread[Thread-23,5,main] tid=10 pid=7)
threadOS: a new thread (thread=Thread[Thread-25,5,main] tid=11 pid=7)
threadOS: a new thread (thread=Thread[Thread-27,5,main] tid=12 pid=7)
Thread[e]: response time = 6000 turnaround time = 6501 execution time = 501
Thread[b]: response time = 2999 turnaround time = 10001 execution time = 7002
Thread[c]: response time = 4000 turnaround time = 21002 execution time = 17002
Thread[a]: response time = 2000 turnaround time = 29004 execution time = 27004
Thread[d]: response time = 5000 turnaround time = 33003 execution time = 28003
```

MFQS:

```
-->l Test2
l Test2
threadOS: a new thread (thread=Thread[Thread-17,5,main] tid=7 pid=0)
threadOS: a new thread (thread=Thread[Thread-19,5,main] tid=8 pid=7)
threadOS: a new thread (thread=Thread[Thread-21,5,main] tid=9 pid=7)
threadOS: a new thread (thread=Thread[Thread-23,5,main] tid=10 pid=7)
threadOS: a new thread (thread=Thread[Thread-25,5,main] tid=11 pid=7)
threadOS: a new thread (thread=Thread[Thread-27,5,main] tid=12 pid=7)
Thread[b]: response time = 1000 turnaround time = 5501 execution time = 4501
Thread[e]: response time = 2500 turnaround time = 8001 execution time = 5501
Thread[c]: response time = 1500 turnaround time = 16004 execution time = 14504
Thread[a]: response time = 499 turnaround time = 24005 execution time = 23506
Thread[d]: response time = 2000 turnaround time = 31006 execution time = 29006
```

Comparison:



The Thread [e] finished second despite having a burst time of 500ms is due to there is a bit of an overhead (quoted from Professor Yusuf Pisan explanation). There for it is put into queue 1, which then it is executed after Thread [b]. Thread [b] finished in queue 1, therefore, it finished before [e] due to the FIFO nature of queue.

As you can see, the Multilevel Feedback Queue Scheduler performs better than the Round Robin Scheduler.

MFQS has a shorter response time. Its quantum is 500ms, which is shorter than the Round Robin 1000ms. This means that it has faster execution and accept the next process.

Shorter turnaround time for MFQS. Short queue allows for faster process. For MFQS, lower priority queue will be executing after higher priority queue. Small process can response, execute, and exit the queue. Also, the multilevel queue allows the process to execute a fixed amount of quantum time before yielding to another process. Compared to the round robin scheduler, fast process needs to wait in the queue for fix (1000ms) even if it's already finishes.

The execution time for MFQS is longer than RR due to the context switch. Some process will be move to lower priority queue, therefore longer execution time.

To summarized, the MFQS beats the RR in response and turnaround time and perform worse in regard to execution time.

Response Time: MFQS > RR due to time slice of 500ms instead of 1000ms.

Turnaround time: MFQS > RR due to small process finish first instead of waiting until all the long process before it to finish.

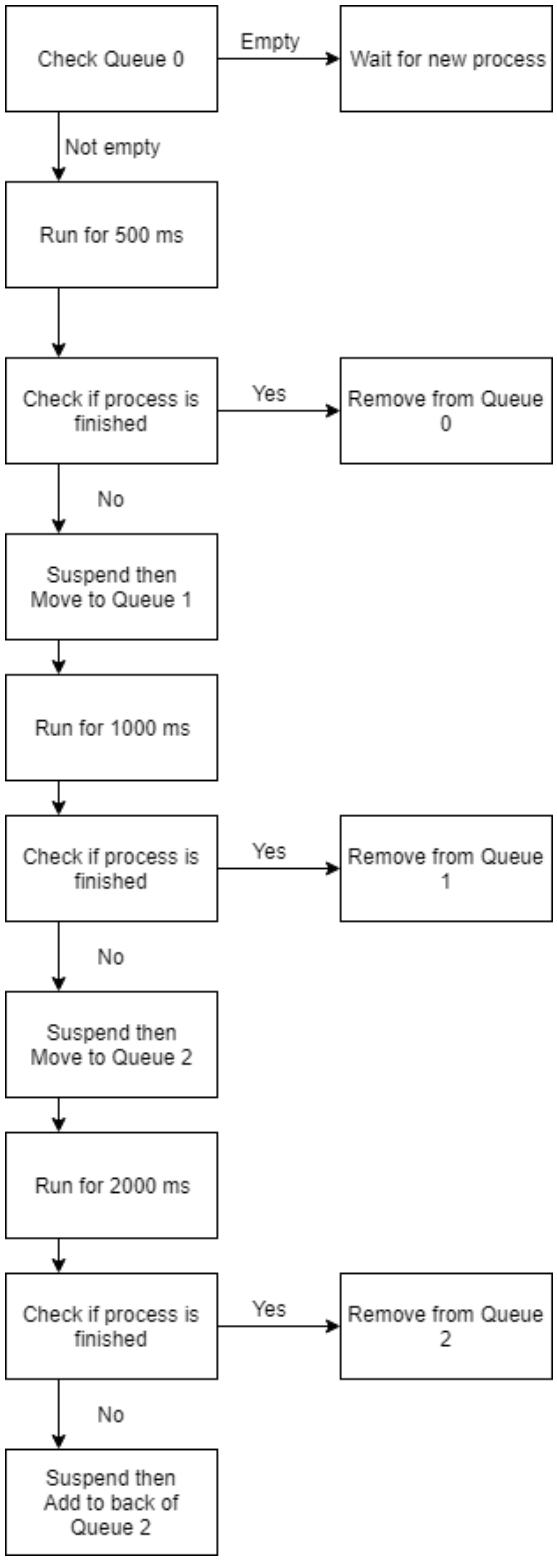
Execution time: RR > MFQS due to context switching and priority handling.

FCFS:
The respond time will be the same, since it also implemented the time slice of 500ms.

The turnaround time will be worse, because if the MFQS implements the FCFS queue for queue 2, it would cost longer to finish. If a thread with long burst process comes first, shorter process comes later will have a longer waiting time to be execute comparing to implementing RR queue for queue 2.

The execution time will be better, since there will be not a lot of context switching for the FCFS implementation. Long process will finish in one run, not having to switch with other process like in the RR implementation. Also, it means that the longer process don't get interrupt, getting to finish uninterrupted.

MFQS Algorithm Discussion:



- For this problem, I modified heavily the run() method. With some minor modification to getMyTcb(), addThread(), Scheduler(). All the other methods stay the same as the RR file.
- For the getMyTcb(), the modification is that there is a for loop to iterate through all three Vector queue. The function is implemented in addThread() and deleteThread(), which is used to add and delete thread from queue. The getMyTcb() is not implemented directly in the run() method, but it is essential part for adding and removing thread from queue
- For the initTid() and Scheduler(), the modification is that there is a for loop to iterate through all three Vector queue and initialize them.
- Modified the Default_Time_Slice to 500ms
- For addThread(), modified so that new thread is always add to queue 0
- For deleteThread(), terminate thread.

For the run() method,

- use three helper method runQ0(), runQ1(), runQ2()
- Will iterate continuously until all process is done
- Check for queue 0 then 1 then 2
- Keep an int quantum recorded how long the thread has run
- Each thread will run 500ms regardless of the queue
- runQ1() and runQ2() have an if statement to check if the process has run 1000ms and 2000ms respectively yet. If so, the thread is removed from the front of the queue and be add in accordance with the method (which will be discuss later on)
- The quantum will be reset to 0 for the next thread
- Check queue 0 for process, always runQ0() if queue 0 size is larger than 0
- runQ1() if queue 0 size is 0 and queue 1 size is larger than 0
- runQ2() if queue 0 and 1 size is 0 and queue 2 size is larger than 0

runQ0() (helper method)

- Get the front TCB, get the first process, start the process and run for 500ms
- If finish, remove the process and return the Tid
- If not, synchronized the queue. then suspend the process
- Remove from queue 0 then add to queue 1

runQ1() (helper method)

- Get the front TCB, get the first process, start the process and run for 500ms
- If finish, remove the process and return the Tid
- If not, check if the process quantum is equal to 1000
- If yes, remove from queue 1 then add to queue 2. Reset quantum to 0
- If not, run once more until quantum is equal to 1000 (runQ1() will run twice for each process before moving that process down to queue 2)
- Check if queue 0 has process, if so change the quantum to 0 so not to mess up the quantum counter

runQ2() (helper method)

- Get the front TCB, get the first process, start the process and run for 500ms
- If finish, remove the process and return the Tid
- If not, check if the process quantum is equal to 2000
- If yes, remove from queue 2 then add to back of queue 2. Reset quantum to 0
- If not, run once more until quantum is equal to 2000 (runQ2() will run 4 times for each process before moving that process to the back of queue 2)
- Check if queue 0 has process, if so change the quantum to 0 so not to mess up the quantum counter