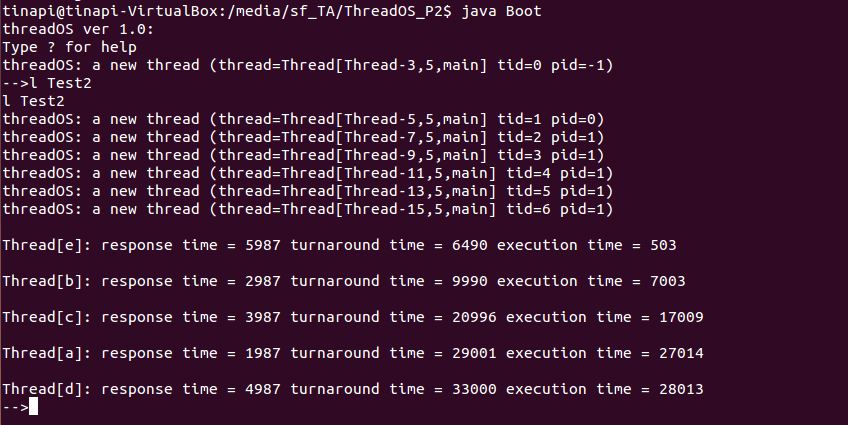
Xueting Pi

CSS 430 Program 2

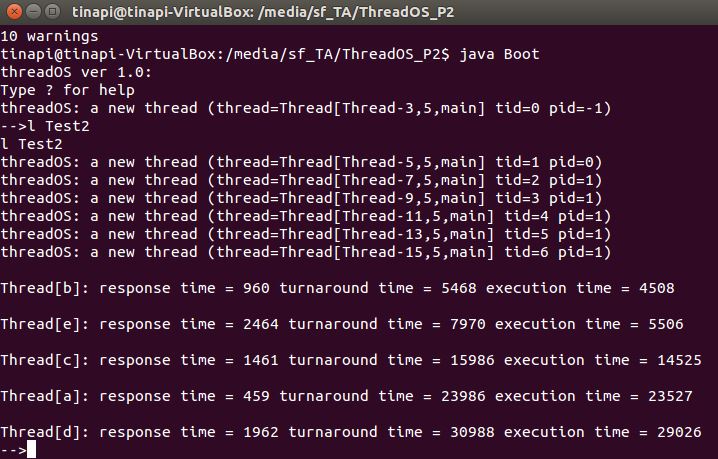
4/29/2015

**Documentation**

Part1:

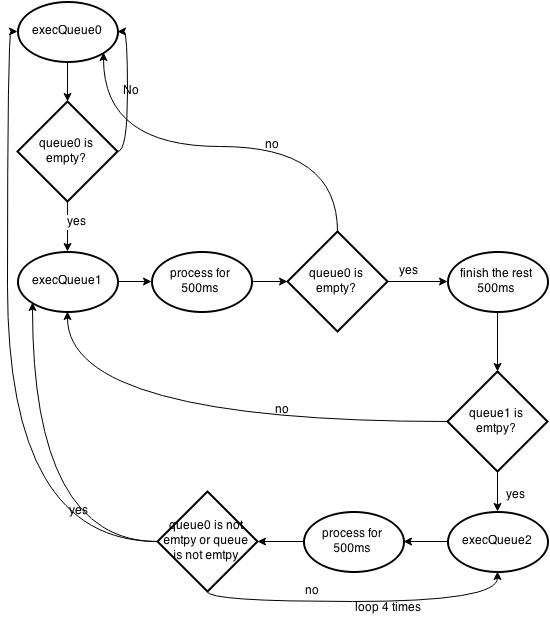


Part2:



Part 2 Algorithm:

First of all, I created 3 queues and wrote 3 methods to process each queue separately. In execQueue0 method, I basically reused the code in original run method, and synchronized queue0 after 500ms quantum. If the current thread is not finished, remove it from queue0 and move it to the tail of queue1. For the method execQueue1, while queue1 is not empty, I process to execute the first element in queue1 for 500ms. After that, check if queue0 is empty, if not, invoke execQueue0 method to run all the threads in queue0 first. When queue0 and queue1 are empty, I can move to queue2. For every 500ms, I would check if queue0 and queue1 are empty, if not, invoke execQueue0 followed by execQueue1 to empty the prior two queues. If the current thread doesn’t finish in 2000ms, move it to the tail of queue2. In my run method, I just keep calling the three queue methods to process.



Test results comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Thread | Response time(ms) | Execution time(ms) | Turn-around time(ms) |
| Round-Robin | a | 1987 | 27014 | 29001 |
| b | 2987 | 7003 | 9990 |
| c | 3987 | 17009 | 20996 |
| d | 4987 | 28013 | 33000 |
| e | 5987 | 503 | 6490 |
| Multilevel feedback queue | a | 459 | 23527 | 23986 |
| b | 960 | 4508 | 5468 |
| c | 1461 | 14525 | 15986 |
| d | 1962 | 29026 | 30988 |
| e | 2464 | 5506 | 7970 |

From the table above, we can tell that generally multilevel feedback queue algorithm has better performance than round robin algorithm in terms of shorter response time and execution time. The reason why multilevel feedback queue algorithm performs better is that by using 3 queues with different priorities, any process that uses too much CPU time would be moved to a lower-priority queue and leaves I/O bound and interactive processes in the higher priority queue. It helps prevent starvation of certain lower priority processes. Round-robin basically puts all the threads in one queue and execute sequentially within the given quantum.

If we were to implement part 2 based on FCFS rather than Round Robin, it would be that we keep processing the first element in the queue until it terminates, then proceed to the next thread. Thread b and thread e will terminate in queue1. Thread a, c, and d will pass to queue2 in order, which doesn’t change. So the expected order when running Test2.java would be b -> e -> a -> c -> d.