Anthony Waddell

CSS 430

Program 2 Report

1. **Clearly explain the design and algorithm for Part 2.**

The design for the algorithm in part 2 was somewhat similar to the algorithm design for Part 1 except with the addition of two more queues and the use of suspend() and resume() calls instead of the setPriority() call to prioritize scheduling. The program FAQ page told me what functions required changing which took out most of the guesswork.

The implementation for Part 2 required 3 Vector queues for threads of different priority so I began by declaring those. With the introduction of three scheduling queues, getMyTCB had to be modified to include returning the thread control block for each of the queues which utilized the same code block from Part 1 but just repeated it for the other two Vector queues. The Scheduler constructors also had to be altered slightly to initialize all three Vector queues. To implement a the MFQ scheduler required that threads begin in the first queue (Q0) so addThread() was changed to always add thread control blocks into that queue.

With those changed made, the bulk of the changes involved the run() method. Since threads start in Q0 and move down to Q2 as the continue to run through their time slices, the first step in run() was to check to see if there were any threads to process in Q0. If there were, they were ran for 500 ms or half of one quantum before getting suspended, removed from Q0, and added into Q1. Threads in Q1 follow the same procedure but since Q1 is not the highest priority, after being ran for their 500 ms, the algorithm looks back to Q0 to see if there are any new threads of higher priority to run. If there are, the algorithm suspends operations in Q1 to return to Q0 and process threads from that queue. If there were not any new threads added to Q0 after the first 500 of running time in Q1, the thread runs again for another 500 ms before checking back to Q0 one more time for any new threads. If so, it will move back to Q0, if not and the current thread has not finished in Q1, it will be suspended, removed from Q1 and placed in Q2, the lowest priority queue.

Threads in Q2 follow a similar procedure as those in Q1 but with a few small differences. They still get to run for 500 ms but instead of pausing to look at Q0 for new, higher priority threads, the algorithm also checks Q1 since it is also higher priority than Q2. To help threads in this queue finish, they run 4 times, with each quantum being 500 ms but pause after their quantum to see if there are any threads to service in the higher priority queues. If threads in this queue still have not finished after their running times, they are suspended, removed from the front of Q2, and placed in the back of Q2 which enforces more of a round-robin policy to help them finish.

1. **Compare the test results between Part 1 and Part 2 using *Test2.java*. Discuss how and why your multilevel feed back queue scheduler has performed better or worse than your round-robin scheduler.**

The multilevel feedback queue scheduler seemed to have performed better response time, turnaround time, and execution time than the round robin style scheduler. Most notably was the response time which was significantly less for the MFQS. The difference in turnaround time was not as significant between the two different scheduler implementations but execution time did display some interesting results. It seems that with the round robin style scheduler that initially thread e had very fast execution time, but the following threads almost looked like they followed a logarithmic growth function. With the MFQS it looks like thread e could no be completed in it’s first burst which resulted in it being placed in a lower priority queue and increased it’s turnaround time over the round robin schedule and allowed thread b to execute just a little quicker.

1. **Consider what would happen if you were to implement the part 2 based on FCFS without pre-emption rather than Round Robin. Your discussion may focus on what happens if you run *Test2.java* in this FCFS based *Queue* (e.g. quantum = 2000 ms). *Note*: It is recommended that you build three different Gantt charts so that your execution performance numbers can clerly be visualized for your discussion.**

If Part 2 were implemented with a FCFS scheduler without pre-emption I would expect to see just about the opposite results that the MFQS scheduler achieved. Execution time would go down and for that reason, it would bring the turnaround time down as well but response time would see an increase since jobs would run until completion.