CSCI 432 Project Writeup, Task scheduler Comparison

Authors:

Devan W. Eastman-Pittam / Ben F. Miller

Problem:

Our problem that we were trying to solve was to see which task scheduling algorithm was the best to use among the; "Shortest Remaining Time", "Round Robin", and the "Priority Based" task scheduling algorithms in different scenarios and in particular had the best TurnAround Time (Time in queue + execution time). We used **2 Main Metrics** and **4 sub metrics** to complete this task; the first 2 are the main metrics and the last 4 are the sub metrics, **First was Average Turnaround Time in milliseconds, next was Average Times Processes Swapped**, then **Average Turnaround Time for Fast,Slow, High Priority, And lastly Low Priority Tasks.**

Hypothesis:

Each metric above is very important for different circumstances, and it is not possible for any algorithm to be the best at all of them. In an ideal scenario, perhaps it would be best to use a mix of all three of these algorithms.

The shortest remaining time algorithm will always work a task till completion unless a shorter task starts, so it should have the lowest average times processes swapped. The turnaround time of the slow tasks should balance the turnaround time of the fast ones, so the average turnaround time should be somewhere near the highest priority algorithm. The average turnaround time for fast tasks should be the best of the three algorithms.

The round robin switches to the next task on each step, so we expect it to have the highest average times processes swapped and the highest turnaround time. It also treats all tasks equally, so it should have a lower average turnaround time for slow tasks than the shortest remaining time algorithm and a lower average turnaround time for low priority tasks than the highest priority algorithm.

The highest priority algorithm will switch tasks each time between all tasks with the same highest priority level. When lower priority tasks are being worked, the scheduler should switch often, but will complete high priority tasks quickly. The average times processes swapped should be better than round-robin and worse than shortest remaining time. The average turnaround time of the high priority tasks should balance out the low priority tasks, so the overall average should be near the shortest remaining time algorithm. This algorithm should have the lowest average turnaround time for high priority tasks.

Input Data:

The input data for the schedulers was in the form of a csv file with 3 data attributes, which were priority, start time, and execution length. There were 6 priority levels that were randomly assigned within a distribution of [1,2,3,4,5,6]. This is normalized to [0.04761905 0.0952381 0.14285714 0.19047619 0.23809524 0.28571429]. The execution length was randomly distributed from 0 to 300 and the start time was randomly distributed from 0 to 500. For the final tests, 20 tasks were generated for each run.

Metrics:

The first main metric was average turnaround time in milliseconds, which was defined as waiting time in the ready queue plus executing time. The second main metric was average times processes swapped, which is the number of times a process is worked on after a different process. Slow tasks are defined as the slowest half of the tasks and fast tasks are the fastest half. High priority is defined as having a priority of three or less, with one being the highest priority, and low priority tasks having a priority of four or greater. We compared average turnaround time for slow, fast, high priority, and low priority tasks.

Setup:

The setup between each scheduler was pretty common, with the only real difference being the implementation of the priority queue. A scheduler was loaded with tasks, then a scheduling was run with it. At the end, the statistics were calculated and printed to StdOut for analysis. While there are uncompleted tasks, a task is extracted from the scheduler, worked, and put back into the scheduler. It is possible that there are no tasks to work at the current time step but there are tasks that haven't started. In that case, no task is worked on that time step. Base statistics are stored on the task object itself for analysis at the end.

Alg1: Shortest Remaining Time

The first algorithm is the Shortest Remaining Time algorithm. It always executes the job with the shortest remaining time. This requires some amount of knowledge about the processes involved as it is not always easy to estimate the amount of time remaining on a task. Because of this restriction, this algorithm is not used very often at the operating system level. It requires a supervisor to estimate the remaining times.

A great situation for this scheduling algorithm would be a web server where most of the requests are processed very quickly and sparsely, leaving time for longer requests to be processed without slowing down the faster requests.

Pros: Prioritizes tasks with low remaining times, so fast jobs finish quickly. Not very computationally hard to calculate the next job if a duration estimate is readily available. This algorithm will work a task to completion very quickly, so fast jobs are not impacted by large queues very much.

Cons: Longer tasks are only worked on when there are no shorter tasks, so longer jobs can take a very long time to complete. It also requires a way to calculate the estimated length, which limits its applications. When the task queue is very large, the long tasks will be in the queue indefinitely.

Alg2: Round-Robin

The second algorithm is the Round-Robin algorithm. It is fairly easy to implement as it doesn't need to know anything about any of the tasks. Each task is placed in a regular queue, worked on, then placed back in the queue. Completion times with algorithms depend only on the number of active tasks. If this algorithm was used in a web server, all requests would be slowed down equally if there was a surge.

Pros: No priorities are taken into account, so all tasks are worked on equally. The execution time is relatively stable for each type of task and is determined by the total number of active tasks. There is no need to calculate priorities for the next task, so computation time is constant and fast.

Cons: Significantly raises individual task time taken to completion with large queues. This algorithm has the least amount of control on order of completion of tasks among all 3 algorithms, so there is no way to make sure any task gets done quickly.

Alg3: Highest Priority

The final algorithm is the priority based task completion algorithm. It functions generally by assigning an integer value as a variable of each task which ranges from 0 (highest priority) to inf (lowest priority). This is the only algorithm of the three that takes into account priorities given by the tasks. Other factors could potentially be used to reevaluate a task's priority dynamically, so this algorithm is the most extensible of the three. As we have implemented it, there is no way for a low priority task to get a higher priority, so it could be on the queue indefinitely.

Pros: This method can quickly implement the tasks deemed the most urgent and prevent their delay in the most efficient method possible. It affords more control to the judgement of which tasks should be done in which order.

Cons: May cause low priority tasks to indefinitely delay completion. If you have many jobs of different priority, the program might be delayed due to time taken to switch to the different tasks. There is no current way to reevaluate a task's priority or finish an almost completed task quickly, which could bring up the average turnaround time.

Results and Analysis:

For a view of the metrics on a per run basis, see the graphics section directly following this. The averages tell most of the story here, but the graphs can be similarly telling.

For our **first Metric of Overall Average TurnAround Time in ms**, the averages were; RoundRobin: taking 1619.1 ms, ShortestRemainingTime: taking 834.25 ms, and PriorityBased: taking 1345 ms.

From here you can see that looking at overall TurnAround time the ShortestRemainingTime alg had the best overall TurnAroundTime being nearly twice as fast as the RoundRobin and PriorityBased Algs.

For our **Second Metric Average Times Processes Swapped**, the averages were; RoundRobin; taking 141.2 swaps, ShortestRemainingTime; taking 71.4 swaps, and PriorityBased: taking 123.3 swaps, showing that the ShortestRemaining Alg swaps the least of all the algorithms.

For our **Third Metric Average Slow Task TurnAround Time in ms**, the averages were ; RoundRobin: taking 2296.9 ms, ShortestRemainingTime: taking 1422.3 ms, and PriorityBased: taking 1441.2 ms. This shows that when handling the Slower tasks IE: the half of the tasks that take longest, the ShortestRemainingTime and PriorityBased algorithms perform roughly equal and the RoundRobin took about twice as long as they did.

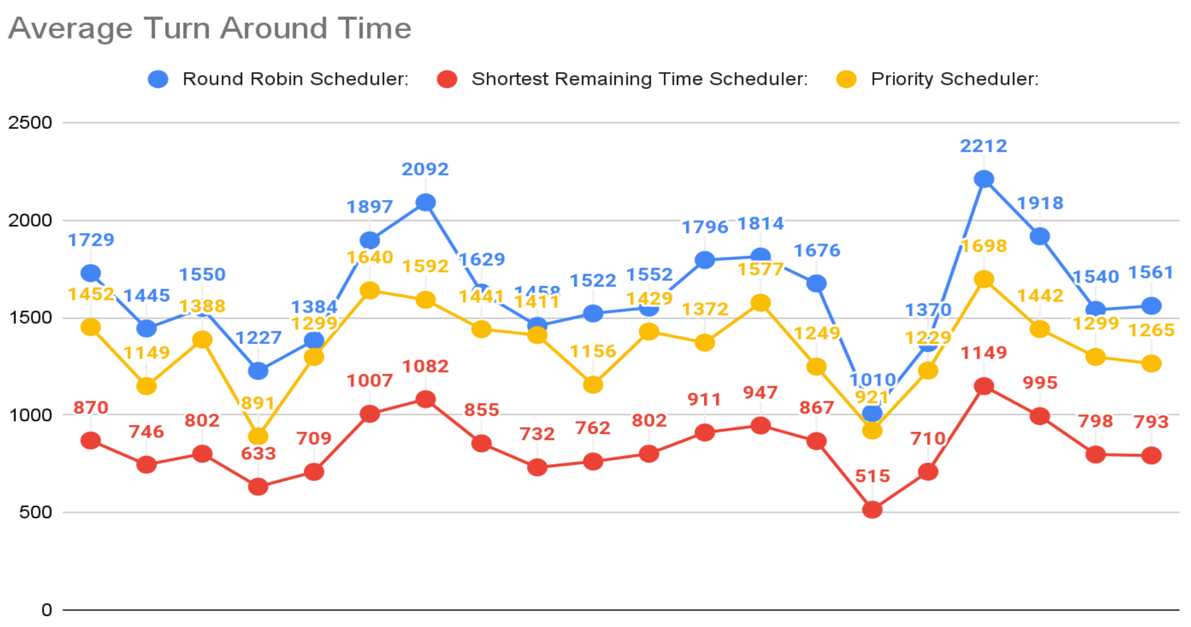
For our **Fourth Metric Average Fast Task TurnAround Time in ms**, the averages were ; RoundRobin: taking 646.55 ms, ShortestRemainingTime: taking 94.5 ms, and PriorityBased: taking 1004.1 ms. From this you can see that the ShortestRemainingTime blew everything out of the water with this one to the point of being up to 10x times faster than the priority based Alg!

For our **Fifth Metric Average High Priority Task TurnAround Time in ms**, the averages were ; RoundRobin: taking 1712.9 ms, ShortestRemainingTime: 875.65 ms, and PriorityBased: taking 487.85 ms.

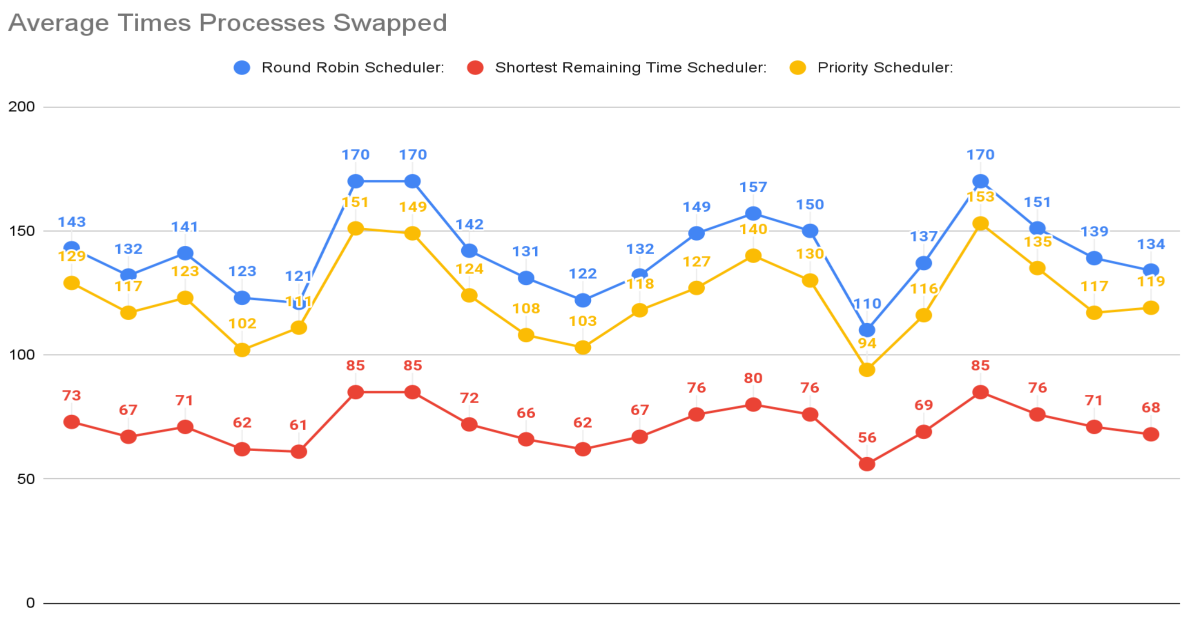
Thus you can see the advantage of the PriorityBased Alg in that if you want things done urgently it's good to use the Priority Alg, but if you want consistency you can use ShortestRemainingTime Alg which consistently performs either best or top 2.

For our **Sixth Metric Average Low Priority Task TurnAround Time in ms**, the averages were ; RoundRobin: taking 1702.55 ms, ShortestRemainingTime: taking 872.9 ms, and PriorityBased: taking 1826.65 ms. So for the Low Priority Tasks ShortestRemainingTime performed best and Priority for once performed the worst.

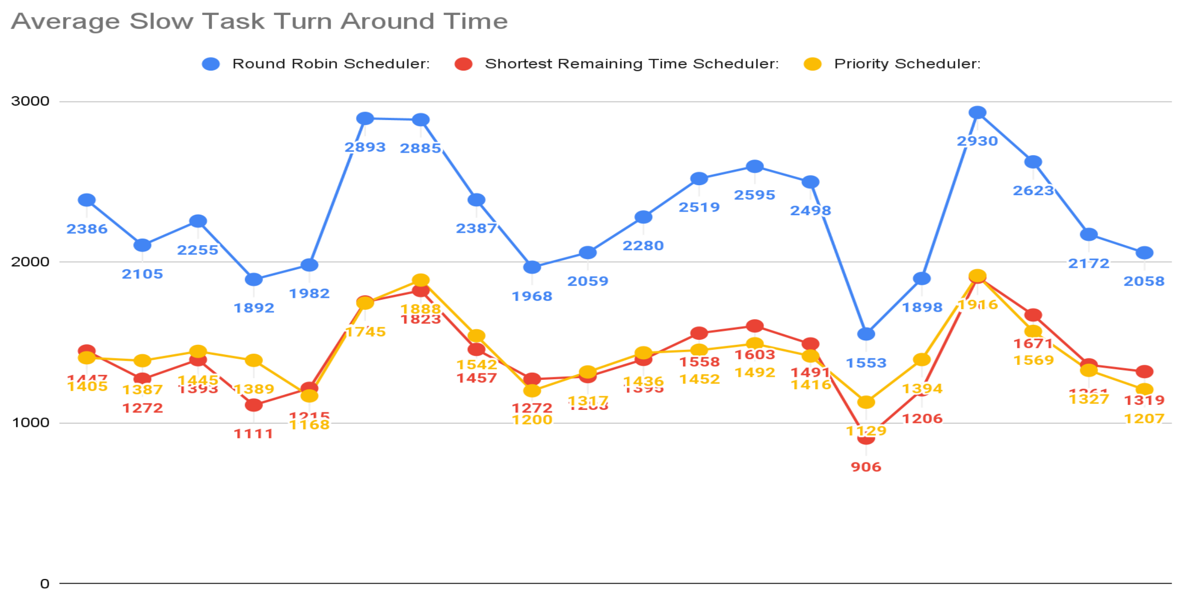
Graphics



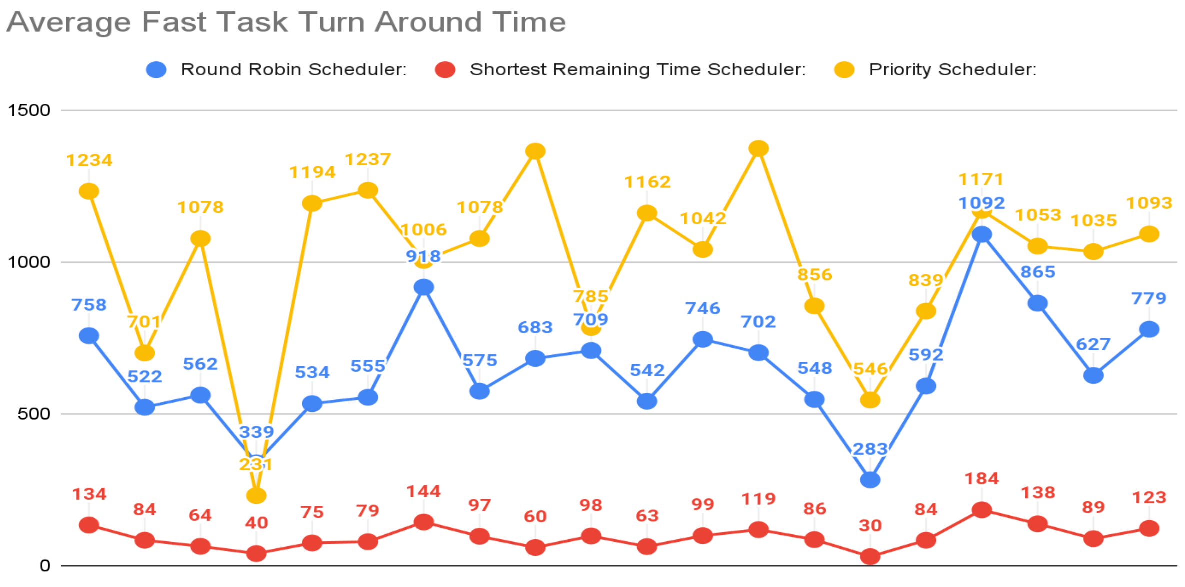
Graph 1.



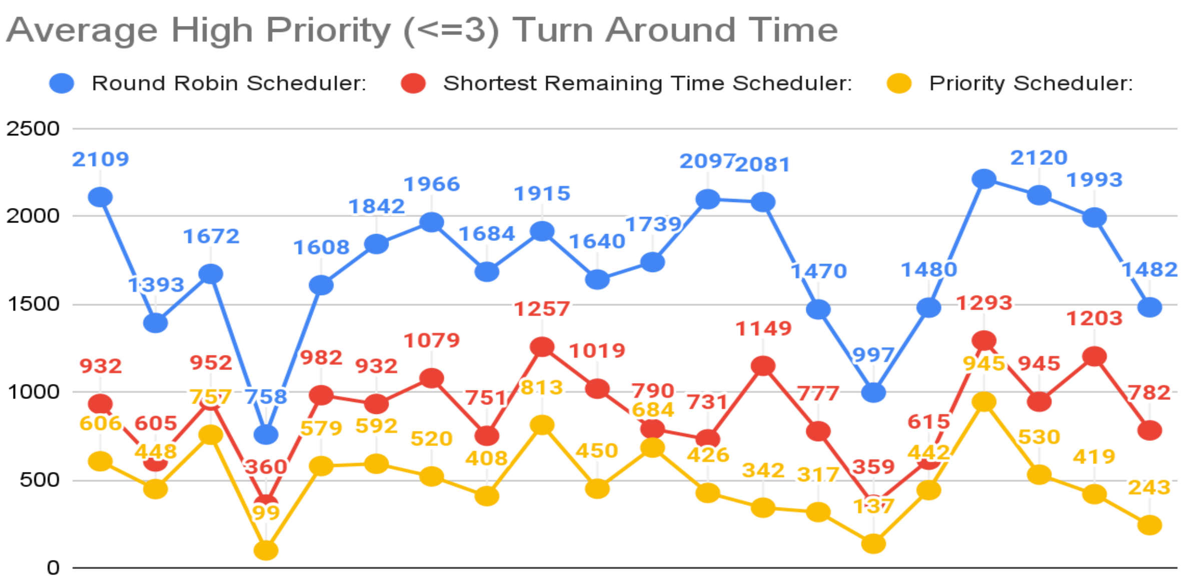
Graph 2.



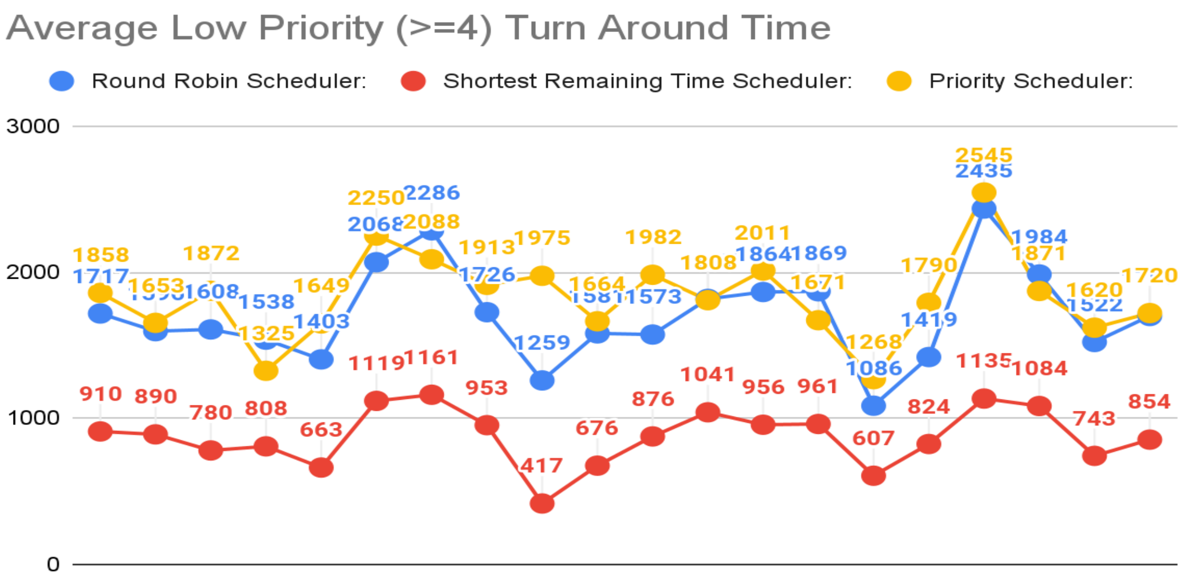
Graph 3.



Graph 4.



Graph 5.



Graph 6.

Conclusion:

Overall, our hypotheses were mostly supported by the data we generated. We did not expect the average turnaround time and average times processes swapped for the highest priority scheduler to be as close to the round-robin scheduler as they are. This does make sense, though, as most of the tasks have a low priority, so working those tasks equally resembles the way the round robin works all tasks equally.

To get a better overall view of the round-robin scheduler, it would have been better to extract statistics with a variety of total tasks. The round-robin scheduler will do well with small task queues, but with 20 tasks with an average length of 150 and an average start time of 300, there were usually many tasks in the queue. We would still expect the shortest remaining time scheduler to have the lowest average times processes swapped and average turnaround time, but round-robin might outperform highest priority with less tasks.

Another surprising aspect of the results was that the average low priority turnaround time for the priority scheduler and round-robin were so similar. Again, this might be different with a different number of total tasks. Perhaps a more surprising result is the similarity between the average turnaround time for slow tasks between highest priority and shortest remaining time. There seems to be no apparent reason for this other than they happened to overlap with the current input data statistics; the total average turnaround time for highest priority balances out slow tasks average turnaround time for shortest remaining.

All three schedulers performed well in different aspects. In building a general system, it would probably be good to have some mechanism to increase priority for a task, but round-robin still performed decently. Shortest remaining time appears to have performed the best on average, but very long turnaround times for long tasks could prove problematic for certain scenarios. In choosing a scheduling algorithm, many factors need to be considered.

References

Os priority scheduling - javatpoint. www.javatpoint.com. (n.d.). Retrieved November 23, 2021, from https://www.javatpoint.com/os-priority-scheduling.

Os Round Robin Scheduling Algorithm - javatpoint. www.javatpoint.com. (n.d.). Retrieved November 23, 2021, from https://www.javatpoint.com/os-round-robin-scheduling-algorithm.

OS SRTF scheduling algorithm - javatpoint. www.javatpoint.com. (n.d.). Retrieved November 23, 2021, from https://www.javatpoint.com/os-srtf-scheduling-algorithm.

Williams, L. (2021, October 6). *CPU scheduling algorithms in operating systems*. Guru99. Retrieved November 23, 2021, from https://www.guru99.com/cpu-scheduling-algorithms.html.

UKEssays. (November 2018). A Comparison of CPU Scheduling. Retrieved from https://www.ukessays.com/essays/computer-science/a-comparison-of-cpu-scheduling.php?vref=1