Rüzgar Ayan 21801984

2020-2021 SPRING CS342 PROJECT 2 Report

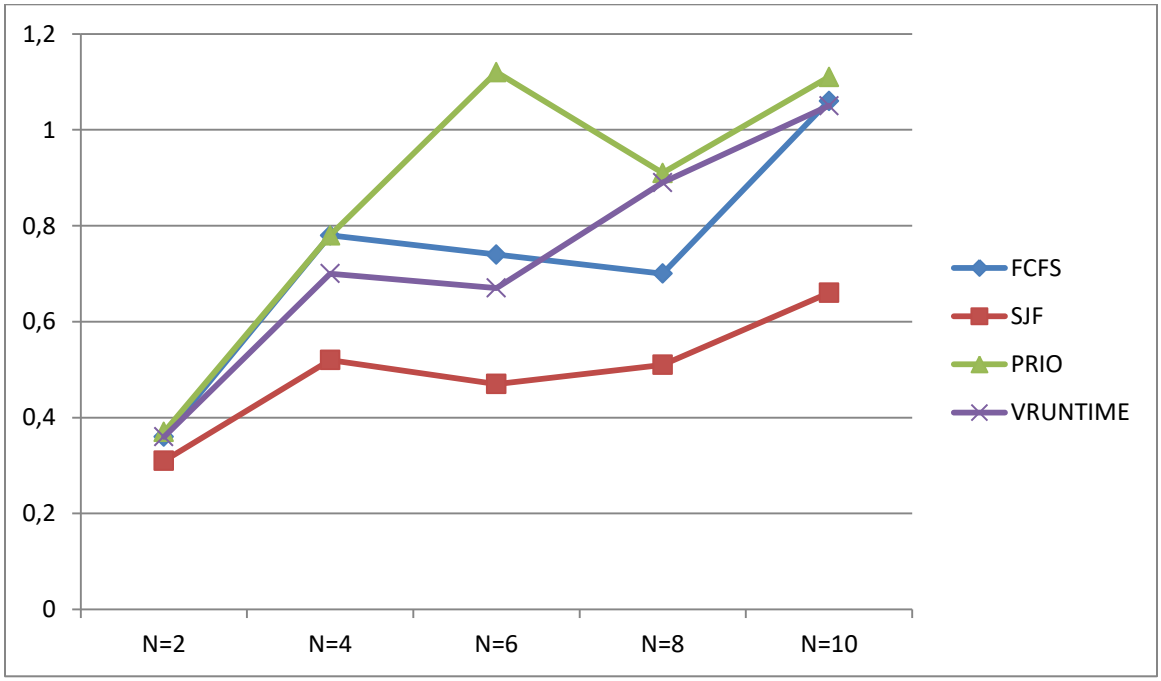
Experiment 1. Average Waiting Times of Different Algorithms

First, I want to compare the 4 algorithms by the average waiting time they cause. For this, I will plot 3 graphs where the values of {minA, avgA, minB, avgB} are chosen different. The horizontal axis on these plots shows the number of threads (N) and the vertical axis shows the average waiting time for a burst in seconds.

First, I set the parameters so that the serving capacity of the cpu is not exceeded. Since I was testing 4 different scheduling algorithms and 5 different thread counts, a total of 20 experiments, I wanted the conditions of these to be as close as possible. So I set Bcount = 120 / N which means there will always be 120 bursts being scheduled. I also scaled the average arrival times with the number of threads. So, the parameters were

minB = 100ms, avgB = 250ms, minA = 100ms, avgA = (400 * N)ms, Bcount = (120 / N)

The plot below shows the results for these parameters



First conclusion we can make is about which algorithm is faster. Definitely from the plot, SJF is the fastest between the four. This is expected as the primary objective of SJF was minimizing the average waiting time. Next, we see PRIO has the largest average waiting time. This is also not surprising

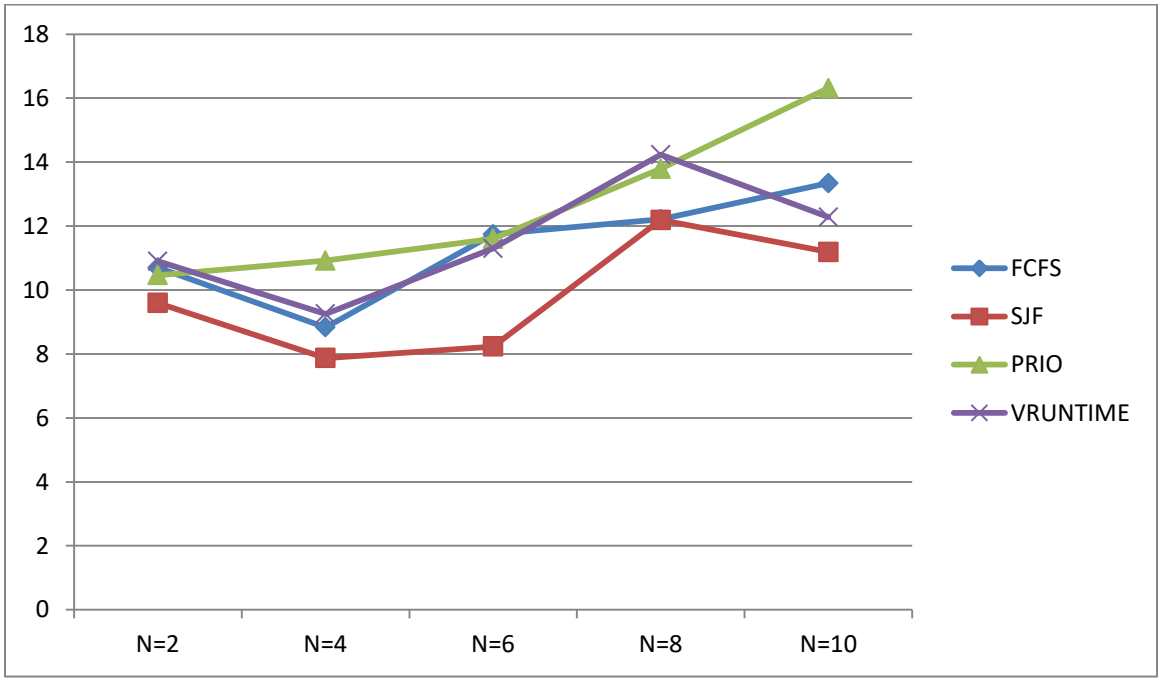
as PRIO only cares about the priority of bursts which has no importance if we are looking at the average waiting time of all the bursts. There is no obvious distinction between the times for FCFS and VRUNTIME. VRUNTIME was expected to be better than PRIO as it is kind of a lighter version of PRIO. The results of FCFS were quite random as it depends only on the order of incoming bursts. Because of this randomness, I run each of these tests 4 times and took their average to plot the above graph.

We can also make comments on the effect of thread numbers above. Although the number of total bursts stays the same and the average arrival times scale with the number of threads in my experiments, it is obvious from the graph that average waiting time increases with the number of threads. This can be interpreted as follows: When there are only 2 threads, the chances that their bursts come on top of each other is low. However when there are 10 threads generating bursts together, the odds that they fill the queue is higher even though arrival times scale up too. This causes higher waiting times as seen in the graph.

Although normally the serving capacity of the cpu shouldn't be exceeded, I changed the parameters that way for the sake of the experiments. I decreased the arrival times and also increased the total number of bursts. In the first experiments, the results were fluctuating a lot between any two runs. But with the following parameters, the results were more stable.

minB = 100ms, avgB = 250ms, minA = 100ms, avgA = (250 * N)ms, Bcount = (240 / N)

The plot below shows the results for these parameters

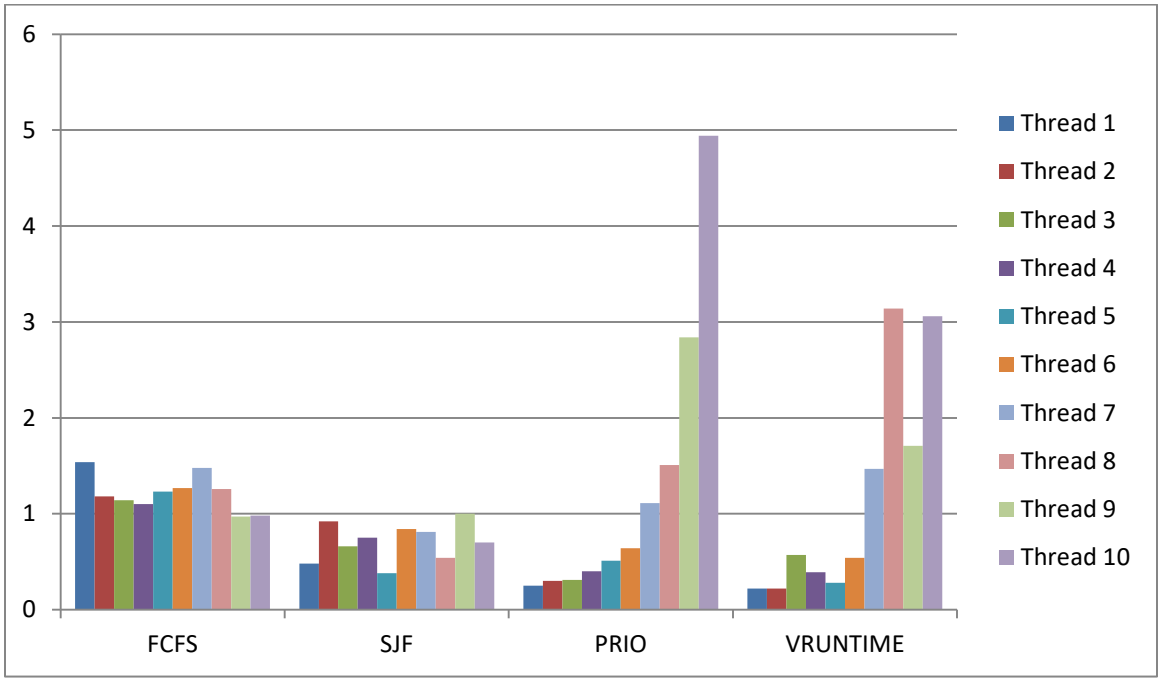


The same conclusions as the before plot can be made for this one too.

Experiment 2. Distrubution of Average Waiting Times of Threads With Different Algorithms

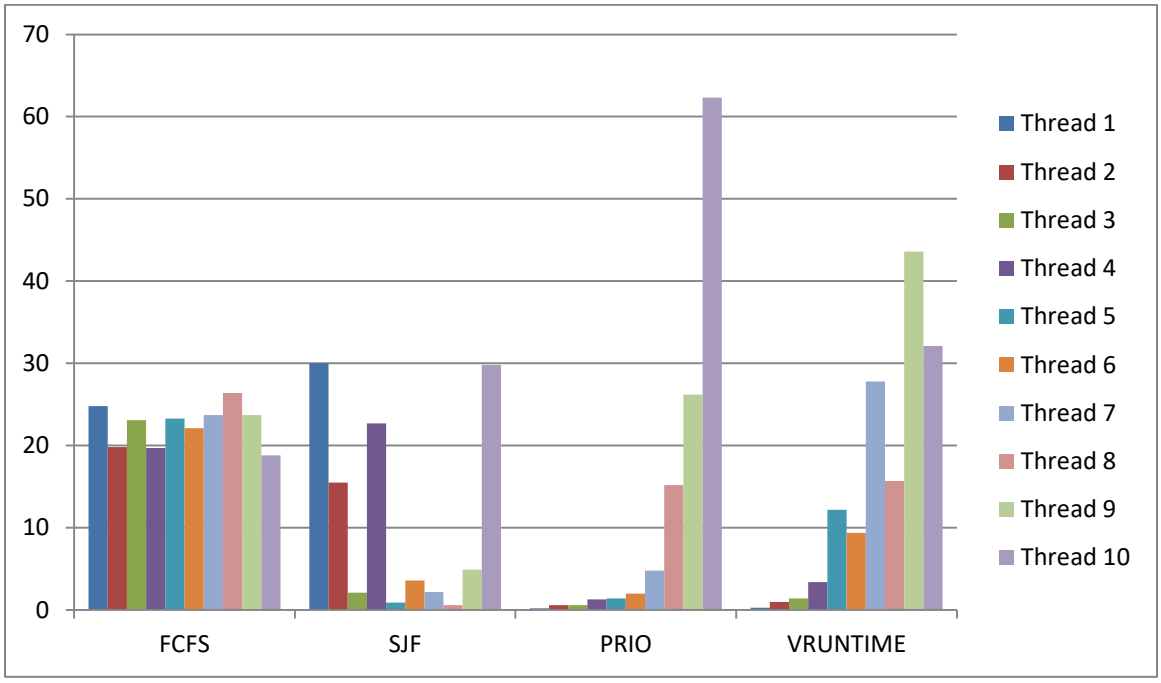
In this experiment we will compare the average waiting times for individual W threads rather than comparing the total waiting time of all threads. Since the aim here is to compare the results thread by thread, I ran all of the following experiments with 10 threads.

minB = 100ms, avgB = 250ms, minA = 100ms, avgA = 4000 ms, Bcount = 100

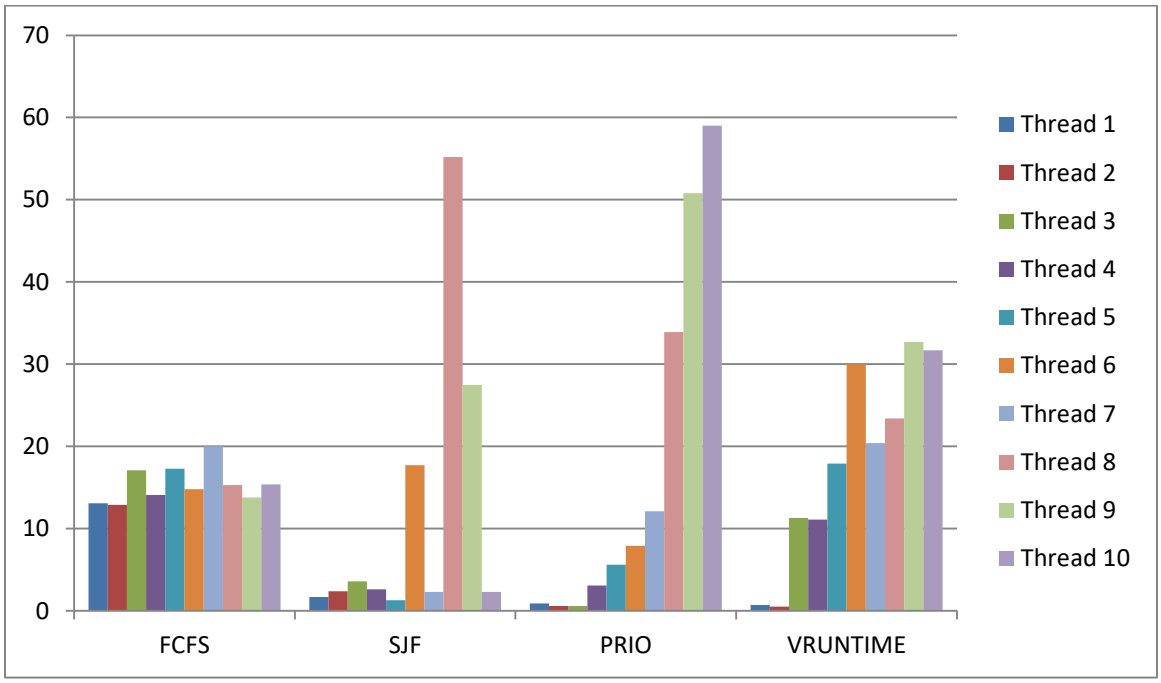


Just like the Experiment 1, I also tried changing the parameters so that the serving rate of cpu is exceeded. That way, I got results that show the characteristics of these scheduling algorithm much more clearly. I have two such experiments and plots below:

minB = 100ms, avgB = 250ms, minA = 100ms, avgA = 2500 ms, Bcount = 100



minB = 400ms, avgB = 500ms, minA = 400ms, avgA = 5000 ms, Bcount = 10



FCFS: There is no distinction between threads. All of them have similar average waiting times and the differences are caused only by the randomness. In that sense, this one is the most fair obviously.

SJF: As seen in the second and third plots more clearly, some random threads have exceedingly higher average waiting times. As I observed from the program outputs, this happens when a thread generates a very large burst by chance. Then, all the bursts generated by this thread must wait for a long time. As we talked in the class, this was one drawback of SJF and methods like aging are exactly used to prevent this.

PRIO: The plot is just as expected. Higher priority threads get their job done almost instantly, whereas the lower priority ones must wait very long times.

VRUNTIME: This can be seen as a fairer version of PRIO. Priority is important here but still lower priority bursts in the queue can be scheduled before the higher priority ones. The plot shows this effect very well.