Operating Systems 1

Assignment 1 Report

Catherine Li
Student Number: LXXCAT004

May 4, 2025

Contents

1 Introduction **1**

2 Approach and Design **1**

3 Experiment Details **2**

3.1 Context Switch Choice 2

3.2 Optimal Time Quantum 2

3.3 Comparing Algorithms 2

4 Results **3**

4.1 CPU Utilization 3

4.2 Waiting Time 3

4.3 Turnaround Time 3

4.4 Response Time 4

4.5 Throughput 4

5 Discussion **5**

5.1 Predictability 5

5.2 Fairness 6

5.3 Possibility of Starvation 6

6 Conclusion **6**

1 Introduction

This report investigates the difference in performance of The First Come First Serve (FCFS), Shortest Job First (SJF) and Round Robin (RR) scheduling algorithm by using the Barman simulation. The algorithms' performance are compared according to their CPU Utilization, Throughput, Turnaround Time, Response Time, Waiting Time, Starvation, Predictability and Fairness.

2 Approach and Design

The given code has been modified to take in additional arguments *experimentNumber* and *findQ* in command line. The former helps differentiate simulations runs based on which seeds are used. The latter is a boolean variable set to True for experiments designed to find the optimal time quantum, q.

Before collecting the data to compare the scheduling algorithms, a suitable context switch is chosen to more accurately reflect reality with the simulation and an optimal time quantum is found through experimentation to best reflect the potential of RR scheduling. When looking for the optimal context switch, only turnaround time is recorded.

Then, CPU Utilization, Throughput, Turnaround Time, Response Time and Waiting Time is measured while the simulation runs. Based on the resulting data , all eight metrics of the algorithms are compared and contrasted.

The data will be summarised as distributions, where instead of bars, lines representing the bars will be utilized to overlay all three algorithms in one graph to better interpretation.

3 Experiment Details

In the Barman simulation, CPU Utilization, Throughput, Turnaround Time, Response Time and Waiting Time are each measured with an instance of *Measure*, stored in *Constants.java* [1]. The shell scripts *quantum.sh* and *algo.sh* were written to streamline data collection. Furthermore, The Jupyter notebook *graph.ipynb*[2] resulted from the experiment is saved in *data*.

3.1 Context Switch Choice

In multi-threaded systems, the kernel simulates parallelism by switching between processes, but each context switch incurs overhead due to the need to save and restore process state.

This overhead is modeled as the cost of switching between drink orders for Sarah in the Barman Simulation. In order to safely switch between drinks, the Barman needs to perform tasks such as put down anything not used for the next drink, clean up any drips on the glass so it doesn't leave an ugly water mark, find the relevant resources for the next drink etc.. Sarah is a skilled Barman (well-written OS System), hence a suitable duration to take to switch between drink orders (context switch) is 4 seconds, 4 milliseconds in the simulation time.

3.2 Optimal Time Quantum

To evaluate RR Scheduling's full potential, it is essential to select the optimal time quantum,q, as this parameter significantly influences the algorithm's performance characteristics. The chosen q should not be too big, otherwise RR scheduling will become FIFO scheduling; and q should not be too small; otherwise the overhead of context switching becomes too expensive.

The shortest drink to make takes 20 seconds, and the longest takes 200 seconds. If q is less than 20, there is more overhead than necessary and if q is over 200, RR becomes FCFS scheduling. Hence, the range of q-values being evaluated spans from 20 to 200, in increments of 10.

The optimal q is chosen based on the shortest average turnaround time. For each q, the simulation is executed 30 times, using 10 unique random seeds, with each seed run 3 times. Each run includes 30 patrons, yielding 30 turnaround time data points per run. The seeds and patron count together allow the average to be taken from a representative dataset. The repeated runs was to make the average more robust and not be skewed by a specific run due to an anomaly. The average is calculated from the turnaround times recorded from all 30 runs and the results are displayed in Figure 1:

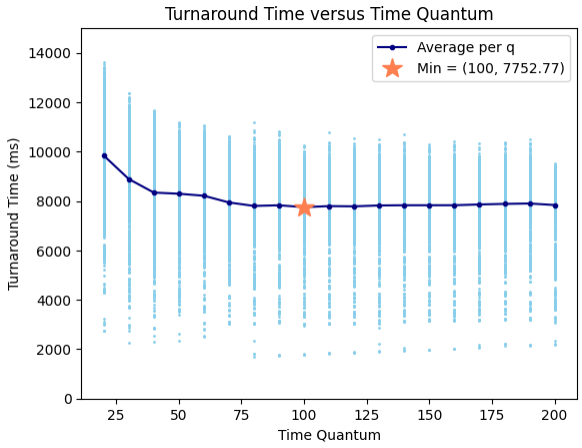

Figure 1: Scatter plot of all the turnaround time data points at different q values. The line graph of the average turnaround time per q is superimposed on top of the data and the best q is marked with a star

The optimal q value is found to be 100[3] from the experiments.

3.3 Comparing Algorithms

For each of the three algorithms, the simulation was executed 180 times, using 30 unique seeds, with each seed run 6 times. CPU utilization results in a single data point per run. To ensure a more representative dataset and capture a broader range of results, the simulation was run more extensively than in the q-value experiments. Refer to section 3 for the summary of data.

[1] Search *Constants* in the provided simulation code will reveal most of the measuring mechanisms the author added.

[2] The notebook can be run to get the graphs, however, the correct path must be defined the notebook. The data is saved in *q_data*, *algo* and *algo_2* respectively. Figure 1 is plotted using *q_data*. CPU Utilization is plotted using data from *algo_2* (as there was an error in *algo's* CPU utilization record) and the other three metrics is plotted using data from *algo*. The file data needs to exist, otherwise when the simulation runs, the program will print a troubleshooting message. This is just the metrics not being able to be record, the simulation runs fine.

[3] This value is close to the rule of thumb, where q should be longer than 80 percent of the CPU bursts. In this simulation, q should be longer than 75

4 Results

This section presents the relevant graphs and draws conclusions for each algorithm based on the visualized data.

4.1 CPU Utilization

CPU Utilization is the proportion of time the CPU is actively working. In the simulation, it is the proportion of time the barman spends making drinks.

The time from which the countdown latch is released till the time barman is packing up is recorded. Thereafter, the total duration of time the barman spends making drinks for each patron is recorded. For each simulation run, the CPU Utilization is calculated based on the total time the barman spent making drinks over the duration in which the barman is a running thread. The results are depicted in figure 2.

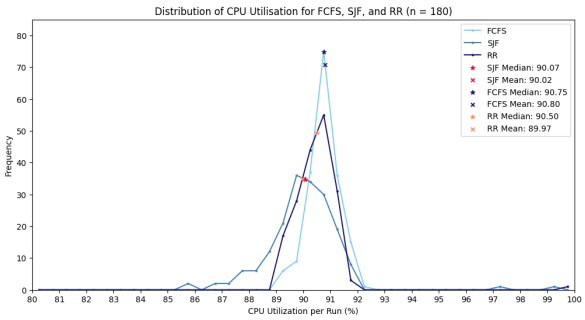

Figure 2: The CPU utilization of the three algorithms is presented, with the median and mean values denoted for each algorithm.

The CPU Utilization of FCFS has the highest average and median value. From the distribution of FCFS's distribution, the CPU Utilizations obtained are also the least variable, because it has the tallest, narrowest distribution curve. The FCFS is the simplest algorithm of the three and requires the least overhead, hence the best cpu utilization.

RR has the second highest average cpu utilisation. The distribution curve closely resembles FCFS, but the values are slightly more variable, evident in the thicker shape of the curve. The average and median value of SJF along with the median of RR closely resemble each other. RR has more context switches compared to FCFS, thus, one would expect its CPU utilization to be less than FCFS, as observed in the data.

SJF has the lowest median value and comes closely second to the lowest mean value. Furthermore, it also has the most variable results as the distribution curve is significantly lower than the other two and it has a greater range of CPU utilization values. SJF is associated with overhead such as maintaining enough data to predict the length of the next CPU burst and maintaining a priority queue, which results in the most overhead and the least CPU Utilization of the three algorithms

4.2 Waiting Time

Waiting time refers to the total amount of time a process has been waiting in a ready queue. In the simulation, this is the total time the patron spent waiting for the barman starts to start preparing their drink. The results are depicted in figure 3.

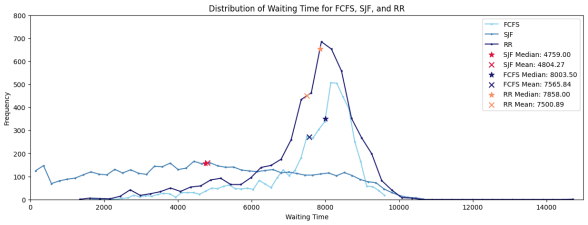

Figure 3: The waiting time per patron of the three algorithms is presented as a distribution, with the median and mean values denoted for each algorithm.

SJF has the shortest average and median waiting time, but the most variable results (distribution is very flat). This reflects the advantage (short waiting time) that the algorithm gains with its extra overhead to process the shortest-next-cpu-burst first. The results are more variable as it is hard to accurately predict which process has submitted the shortest job.

RR has the second shortest average and median, while FCFS has the longest average and median. The distribution of FCFS and RR closely resemble each other. FCFS revealed to have the longest waiting time because processes can be stuck behind processes that need long cpu bursts. RR defines a time quantum that lets processes take turns with the CPU. This perhaps contributed to a slightly shorter average and a negligibly shorter median, where threads are less vulnerable to the convoy effect (e.g. patrons waiting behind a B52 drink order).

4.3 Turnaround Time

Turnaround Time is the total amount of time for a process to complete since its submission. In the

simulation, the total time from when a patron submits their first drink order to when the patrons imbibe their last drink is recorded. The results are depicted in figure 4.

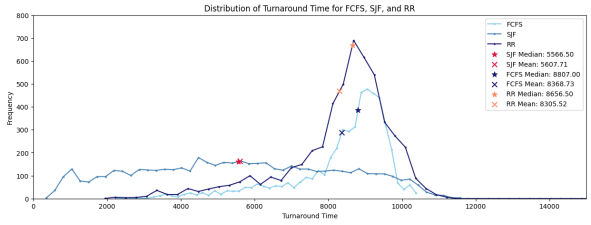

Figure 4: The turnaround time per patron of the three algorithms is presented as a distribution, with the median and mean values denoted for each algorithm.

The results are nearly identical to waiting time. SJF holds the shortest average and median turnaround time. RR has the second shortest average and median, while FCFS has the longest for the same reasons discussed in section 4.2.

SJF appears more evenly distributed while the other two have a distinct spike. This means SJF also has the most variable results of the three algorithms.

4.4 Response Time

Response Time is the time between when a process is submitted and when the first response is produced. In the simulation, the recorded time is the time between when the first drink order is placed to when each patron receive their first drink. The results are depicted in figure 5.

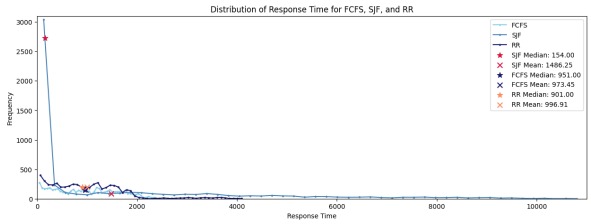

Figure 5: The turnaround time per patron of the three algorithms is presented as a distribution, with the median and mean values denoted for each algorithm.

SJF has the longest mean response time, followed by FCFS, and RR, respectively.The mean aligns with the theoretical expectations established in the course, where SJF has better average waiting and turnaround time, while RR has better average response time (due to the limited q given to each process at a time).

The relative behaviour of the median RR and FCFS response time is similar to the mean value. The distribution of both algorithms are also relatively similar. However, the median value of SJF differs from the expectations of the mean. SJF has the shortest median response time, followed by RR and lastly FCFS. The median of SJF is small due to the relatively large amount of short response times observed. The mean of SJF is skewed to the right of the median by the long right tail of data points representing extremely long response times, relatively. This may be the result of giving less CPU-hungry processes priority and more short drinks are attended to first, while patrons who order a longer drink at first will wait significantly longer (evident in the long tail).

4.5 Throughput

Throughput is the number of processes completed per unit time. Similar to CPU Utilization, the start time (from the release of the countdown latch in *Barman.java*) and end time (till when the barman packs up) is recorded. However, the end time of each patron process is record in place of CPU Utilization intervals. Throughput is calculated by dividing the total duration of the barman being operation (end time - start time) into 100 windows [4]. The results are displayed in figures 6, 7, and 8.

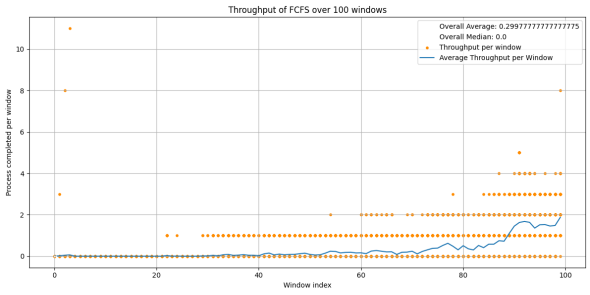

Figure 6: The throughput of FCFS scattered at each of the 100 windows. The average throughput at each window is also included. The overall median and mean values are denoted in the legend.

For all three algorithms, the average throughput per window is roughly 0.3, and the median is 0. The

[4] I have experimented with a sliding window, where the window size and step to slide each window is fixed. However, some simulation runs were significantly longer than others due to the seed used (as excluding different seeds, such as the one for experiments labeled 2, helped mitigate the problem).These simulation runs resulted in significantly larger amounts of windows. This resulted in the data points being concentrated narrowly on the two sides of the graph, making the general behaviour of throughput hard to interpret. Hence, i have divided up the program, execution into 100 non-overlapping windows to standardize the amount of windows for better visualization.

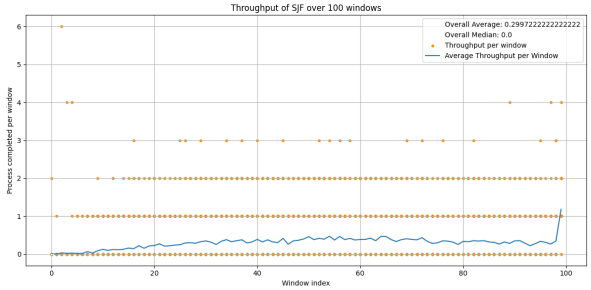

Figure 7: The throughput of SJF scattered at each of the 100 windows. The average throughput at each window is also included. The overall median and mean values are denoted in the legend.

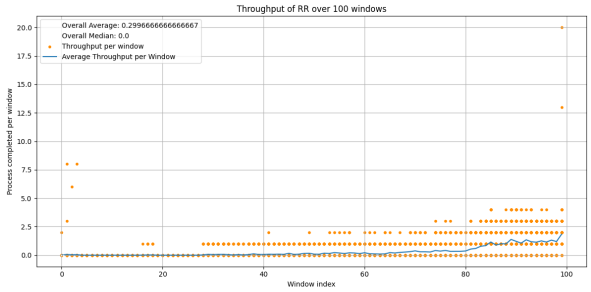

Figure 8: The throughput of RR scattered at each of the 100 windows. The average throughput at each window is also included. The overall median and mean values are denoted in the legend.

low average throughput is contributed the amount of non-sliding windows[5] relative to the number of patrons per simulation run (30 patrons that finish per run.). The median value is an indicator that over half the windows in the simulation runs did not have processes finish in them. This reflect the spike of average throughput at later windows (especially for window 100 for all three algorithms), indicating the concentration of processes finishing towards the end.

The distribution of SJF has the flattest distribution, which means it's throughput is the most consistent of the three algorithms. Less CPU-hungry processes finish earlier whilst the others finish later. The distribution of FCFS and RR are similar in the sense that they both have the majority of the processes finish towards the end. Additionally, RR has some runs that have relatively large throughput at the end (e.g. 20) meaning most patrons have requested a selected of drinks that take a similar time to make and imbibe. This causes most of the processes to finish at the end of the simulation, around similar times.

5 Discussion

The Predictability, Fairness, and possibility of Starvation will be discussed based on the above five observed metrics.

5.1 Predictability

Predictability is how variable the algorithms perform across the five metrics presented in section 4. A variable performance is identified using the distributions through metrics that have lower spikes and thicker tails (or appear more uniform in height) and has a larger range of observed values. A predictable performance is evidenced by the clustering of the majority of data points within a relatively narrow range, resulting in a tall spike in the graph, with a smaller range of observed values.

SJF has low overall predictability. In figure 2 (CPU Utlization), 3 (waiting time), 4 (turnaround time), and 7 (throughput), the SJF distribution curve is significantly more flat[6] compared to the other two algorithms as well indicating high variability in performance (thus low predictability in performance).

However, SJF exhibits high predictability for short processes, as they will be scheduled to run on the CPU first, and low predictability of longer processes as their execution on the CPU can be indefinitely delayed due to shorter processes arriving. This is evident in figure 5 (response time), where there is a large concentration of patrons that experienced a short response time, but the distribution curve flattens out to a long tail reflecting the high variability in response time for patrons that don't have the shortest drink order.

FCFS is moderately better in predictability than SJF, where the processes are scheduled to run on the CPU in the order they arrive in. The minimal level of overhead (compared to RR and SJF) allows it to have a more predictable CPU Utilization, evident in the tall and narrow curve of FCFS in figure 2. The distribution curves for FCFS in Figures 3, 4, and 5 demonstrate a moderate degree of predictability in comparison to the other two algorithms. This is shown by most of the data points

[5] sliding window will have a more interesting visualization and mean/median value, but the issues associated with it(discussed in footnote 2) were too detrimental. The average at each window index reflects the behaviour of the algorithms sufficiently for analysis

[6] The turnaround time and waiting time response curves are unlike the the response time distribution curve. There is no spike on the left of turnaround time and waiting time, because they are measured based on the combined result of five drinks, not just one.

being grouped closely together at the middle or left hand side of the graph. In the case of turnaround time and waiting time, this concentration leads to a noticeable but moderate peak in the distribution. Based on the concentration of orange dots on the left in figure 6 compared to figures 7 and 8, FCFS has a similar level of throughput predictability as RR, and better predictability than RR.

The unpredictability of FCFS comes from the algorithms dependency on the order in which the request for CPU Utilization is made. This is evident in the tails of the graphs mentioned above.

RR displays moderate predictability in terms of its execution patterns of assigning each process a fixed q on the CPU at a time. The rough waiting time each process can also be estimated relatively simply by multiplying the amount of processes in from of the current process with the predefined q.

From the graphs, RR exhibits a similar behaviour for predictability across the five metrics compared to FCFS. This is likely because the q chosen (100) is higher than 93.3 percent of time reuqired to make drink (cpu burst). Hence, the RR scheudling exhibits behaviours similar to FCFS most of the time.

5.2 Fairness

SJF is the least fair out of the three algorithms as shorter processes are favoured. Long processes may need to wait indefinitely (or till the end of the simulation) have CPU time. This is evident in figure 5 (response time), where the difference between the average response time and median response time is very large, indicating some processes being favoured, resulting in very short response times for them.

FCFS is moderately fair, relatively, as processes are assigned CPU time in the order they arrive. However, shorter processes may be stuck behind longer processes, leading to the convoy effect. The long processes could hog the CPU resulting in slight unfairness.

RR ranks high in fairness, as each process gets an equal share of CPU time in the order of their arrival. The algorithm ensures that the CPU is not monopolized by a single thread like FCFS, mitigating the convoy effect.

5.3 Possibility of Starvation

SJF is the most prone to starvation for the same reasons that makes it the least fair. In a scenario where the CPU runs indefinitely with continuous cpu utilization requests, long processes may be indefinitely postponed, leading to starvation. In the simulation, longer process requests are only processed towards the closing of the bar. This is evident in the range of response times in figure 5 compared to the other two algorithms. Despite running the same set of experiments multiple times across the algorithms, the right tail of the response time curve for SJF exhibit data points with extremely high response times, evident of starvation.

For FCFS, starvation will only occur if a process with an infinite loop monopolizes the CPU. This is a rare event and despite the convoy effect (which may give the illusion of starvation) every process is guaranteed to execute eventually. Thus, the possibility of starvation is low.

Similarly for RR, all process will eventually be given CPU access in a cyclic order and the probability of starvation is low. The processes scheduled with the RR algorithm, thus, theoretically, is even less likely to starve compared to FCFS, in the edge case of an infinite loop.

From the observed data in figure 3,4, and 5, waiting time, turnaround time and response time of both FCFS and RR don't exhibit any large outliers values, which confirm the theoretical points discussed above.

6 Conclusion

SJF exhibits the best performance in terms of average waiting time, turnaround time, response time for short processes and throughput. It does not have good predictability, fairness and CPU utilization with high possibilities of starvation.

FCFS is simple, fair, moderately predictable, and starvation-free most of the time. It has the worst mean and median waiting time, turnaround time, likely due to the convoy effect. In terms of responsiveness, FCFS and RR perform similarly.

RR exhibits similar performance to FCFS with a better median response time and overall performance on waiting and turnaround time. This algorithm is also fair, moderately predictable and starvation free.

Hence, the recommended scheduling algorithm recommended for Sarah is RR. SJF is the least appropriate as it is the least fair and the scenario is not starvation-tolerant as angry patrons can write bad reviews that tarnish the reputation of the bar. Furthermore, it is important to keep the patrons happy by having good responsiveness (and RR has out performed FCFS in turns of turnaround time as well). Although more time may be spent switching between making drinks like B52, most drinks will not need a context switch and it is important to make sure customers receive their first drink fast (by mitigating the convoy effect created by drinks like B52) so they are less likely to change bars.