

CSC3002F Assignment 2 – Process Scheduler Sarah the Barman

KSHKAV001

1. Project Overview

The objective of this project is to simulate and evaluate various CPU scheduling algorithms – First-Come-First Serve (FCFS), Shortest Job First (SJF) – non-pre-emptive and Round-Robin (RR). The algorithms are simulated in the context of a barman which acts as the CPU who makes drinks for patrons where each patron is treated as a separate process with a fixed number of jobs(drinks). The performance of the specified algorithms is evaluated based on various metrics including response time, waiting time, turnaround time, CPU utilisation time and throughput.

***Note: Every SJF is mentioned it refers to non-pre-emptive SJF only in the entire report.**

2. Solution Implementation

The solution was implemented in java by using `System.currentTimeMillis()` added to the program to get timing values for each process (patrons) which are further summarised to get mean, median and distribution. In the context of this simulation timing values are recorded as described in Table 1. Each drinks has a fixed preparation time which is taken as “CPU bursts” and each patron orders a fixed number of five drinks.

Performance metric	Meaning in this simulation
Response Time	Time from when a patron places the first drink order until the first order is received by the patron.
Waiting Time	Time from when a patron places a drink order until the barman starts preparing the order. This is taken as a total for all 5 drinks ordered the patron.
Turnaround Time	Time from when a patron places the first order until the patron is finished drinking the last drink.
CPU Utilisation	Time the CPU (barman) spends on preparing drinks as a fraction of the total simulation time.
Throughput	The number of processes completed (patrons) per fixed unit time (taken as 2000ms for this assignment).

Code description

- In the Patron.java three arrays are implemented to calculate times of order placed, order received and order drink complete. This is useful in calculating the response time, waiting time and turnaround time and log it in the TimingLog.java. All Patrons finish times are recorded in TimingLog.java for throughput calculations.
- In Barman.java the time when a barman starts working on a job till the job is complete is recorded – this is used to calculate the total time the barman spent preparing drinks within the TimingLog.java for CPU Utilisation.
- All processing for the metrics is done in the TimingLog.java. It records the total simulation time, all performance metrics for each patron in experiments.csv and summary values across each run in experiments_summary.csv. The throughput is recorded by sorting the

collection of patron finish times and recording them over a window of 2000ms giving approximately 10 equal time intervals.

- All times are considered in milliseconds.

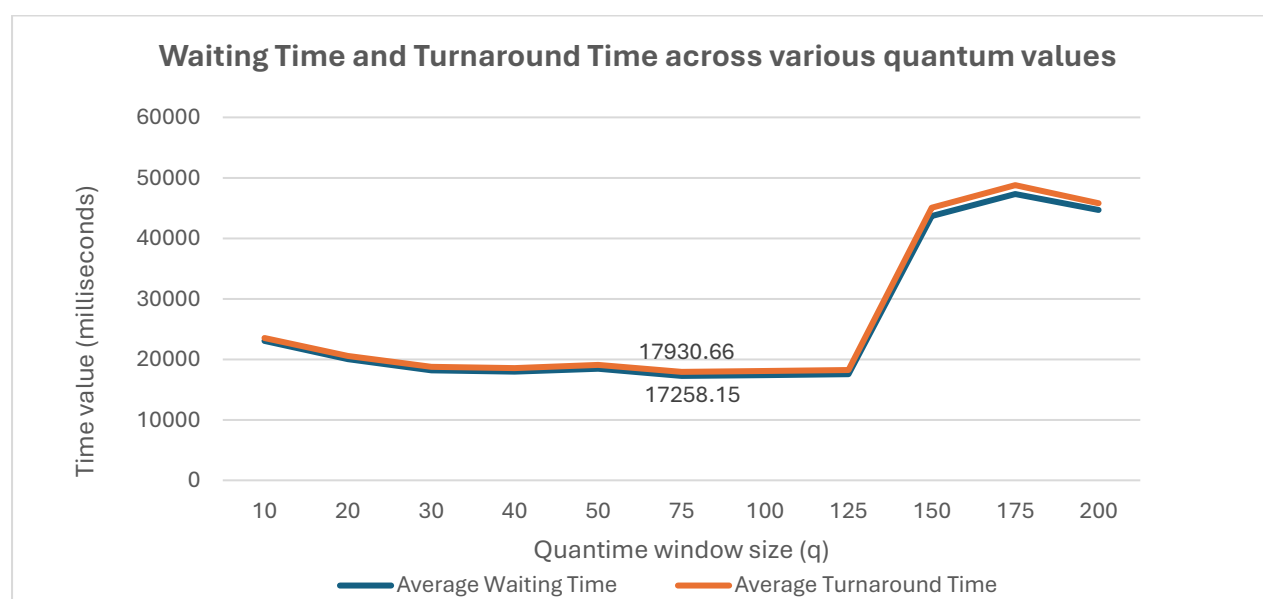
3. Context Switching Time

The context switching time is the time spent between switching the processes (patrons) by the barman. This time is included in the total execution time as it represents the transition overhead of process switching. In a real-world simulation it would take a barman a few seconds to clean up between making drinks. Ideal context switching time is zero within an operating system. In this experiment context switching time, c , is taken as 3 based on the following reasoning:

- It shows a value that realistically simulates the switching time in an operating system. Taking a time of zero would be too idealistic and taking a real-life context switching time would be meaningless as we are working with process scheduling with a CPU.
- A very small value such as 1 would make context switching time very insignificant and a very large value would unrealistically penalise algorithms like Round Robin from an operating system perspective.
- Through initial trial runs, we observed that $c = 3$ provided a reasonable balance.

4. Quantum Values for Round Robin Scheduling Algorithm

The round robin scheduling algorithm is highly dependent on the scheduling time quantum q which determines the maximum amount of time a patron can be served before the next patron is given a chance to be processes. To determine optimal values the RR algorithm was run with the same seed across different time quantum values ranging from 10 to 200. This range is based on the CPU burst time (preparation time of drinks) which ranges from 20 to 200. The best quantum value is considered as the one which results in at least 80 percent of the CPU bursts to be shorter than the time quantum. This experiment is done using `q_experiments.sh`. The optimal value for time quantum, $q = 75$ based on Graph 1 shown below. The value is chosen as it results in the lowest average turnaround time and lowest average waiting time per patron. **The number of patrons selected was fixed at 80 patrons.**



As can be seen from the graph and the data experiment the lowest average waiting time and turnaround time occurs at $q=75$. The lowest average waiting time = 17258.15 ms and lowest average turnaround time = 17930.66 ms. 73% (11 out of 15) drinks have a preparation time shorter than the time quantum and approximately 87% drinks fall within the time quantum (with prep time = 75).

5. Scheduling Algorithm Comparison

Multiple experiments were conducted to effectively compare the performance of the three scheduling algorithms across various metrics discussed in the section below. As per the sections mentioned above, the following values were fixed as:

- Number of patrons = 80 patrons which gives thorough results and accounts for thorough experimentation without overloading the system.
- Context switching time (c) = 3
- Quantum value (q) = 75

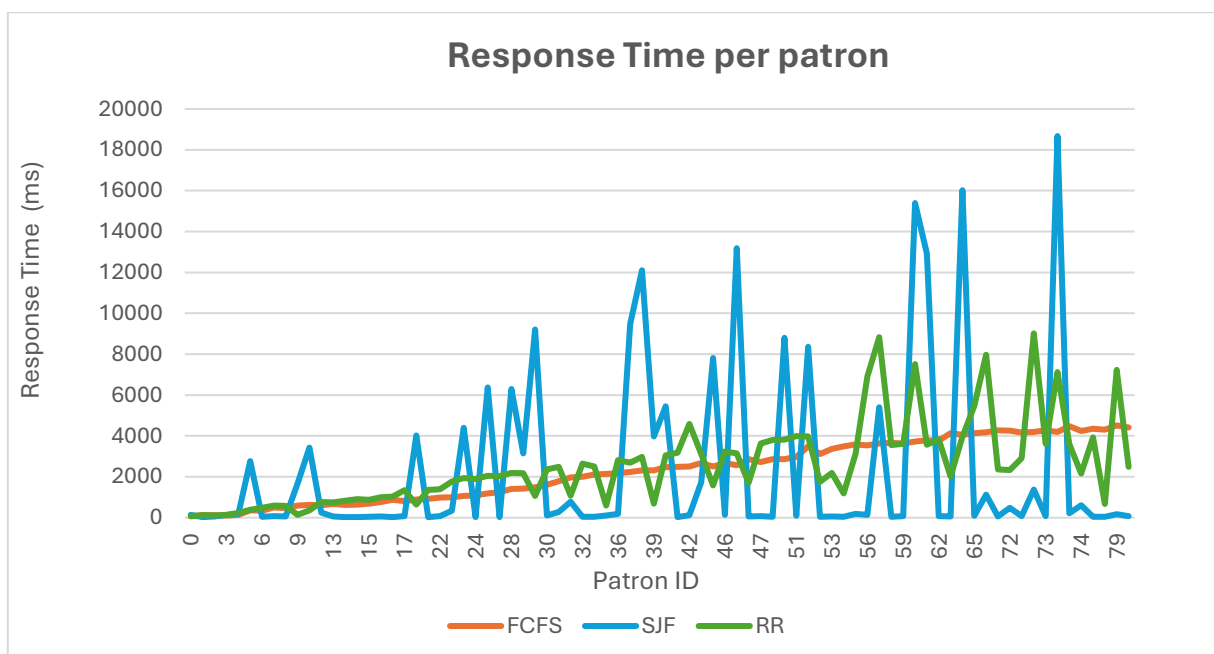
While maintaining these fixed values the following experiments were conducted:

Experiment 1: Run the experiment with a constant seed (101 in this case) for all three scheduling algorithms. This helps in obtaining data across the different patrons in each case. CPU utilisation and Throughput is calculated for each algorithm.

Experiment 2: This experiment runs by conducting experiment 1 across 10 different seeds. In each run the average values of the waiting time, response time, turnaround time, CPU utilisation and throughput are calculated. This is completed using `run_experiments.sh`

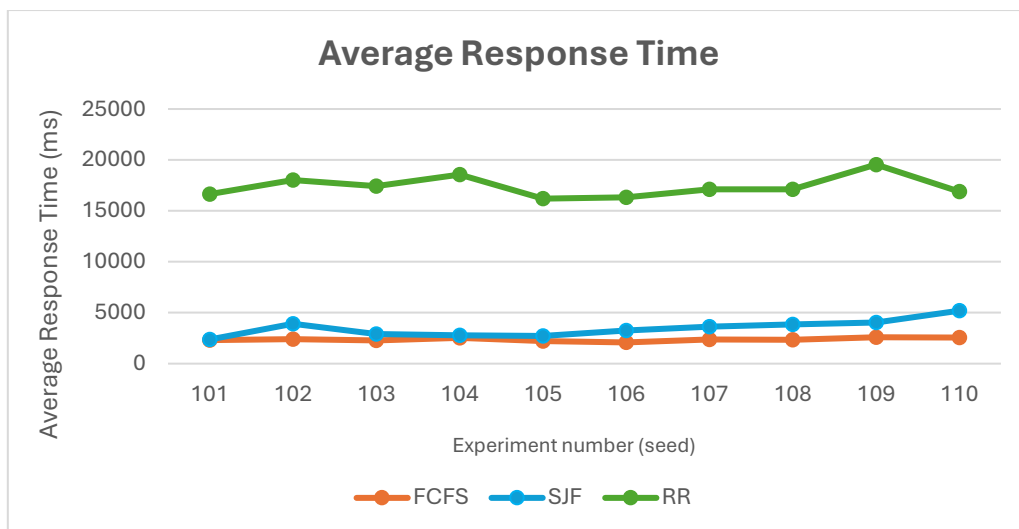
The results obtained for both experiments are analysed per metric as discussed below.

5.1 Response Time



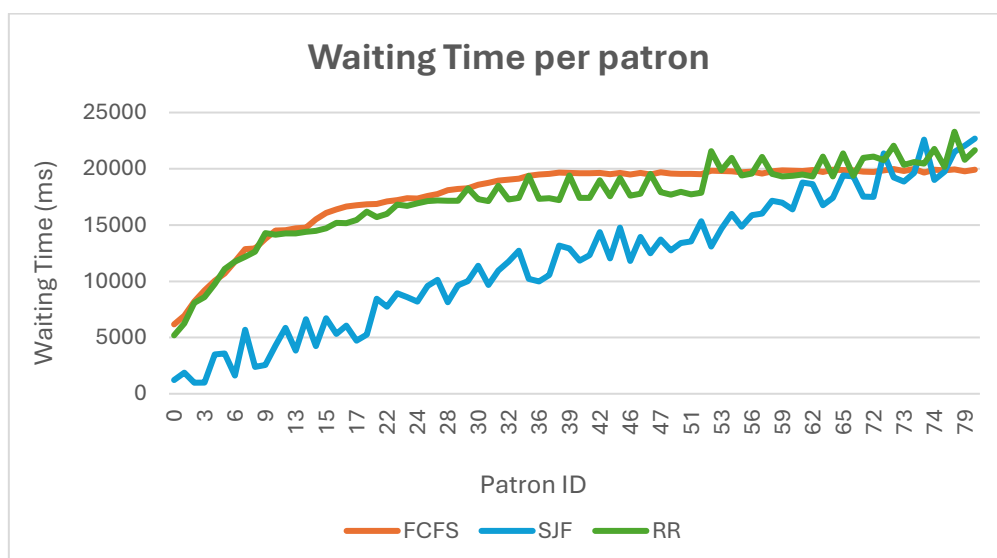
Scheduling Algorithm	Average Time (ms)	Median Time (ms)	Variance (ms)
FCFS	2320	79	2043312.65
SJF	2371	59	18668360.12
RR	2572	79	4222792.53

- As expected SJF has the lowest median response time and one of the lowest average response times along with FCFS. However, it is important to know that the response time varies a lot (highest variation) – as patrons who request drinks with very high execution time can be served much later as the drinks with the shortest execution time are prioritised.
- RR despite offering time slices access to all jobs exhibits higher average response times due to repeated context switches and the time to cycle through to each job. The variance is lower than SJF and is relatively small but higher than FCFS.
- FCFS has the lowest average response time but a very high median as the response time increases significantly for patrons with higher patron IDs i.e. much later processes.
- While RR does have the highest average response time it has the lowest median response time and a relatively lower variance. In the context of a barman situation, it is important to have less variation in response time across all patrons.



- The averages taken across the different seed confirm the results discussed above. Overall, the algorithms FCFS and SJF have lower response time but high variances and medians making RR the most suitable for this metric.

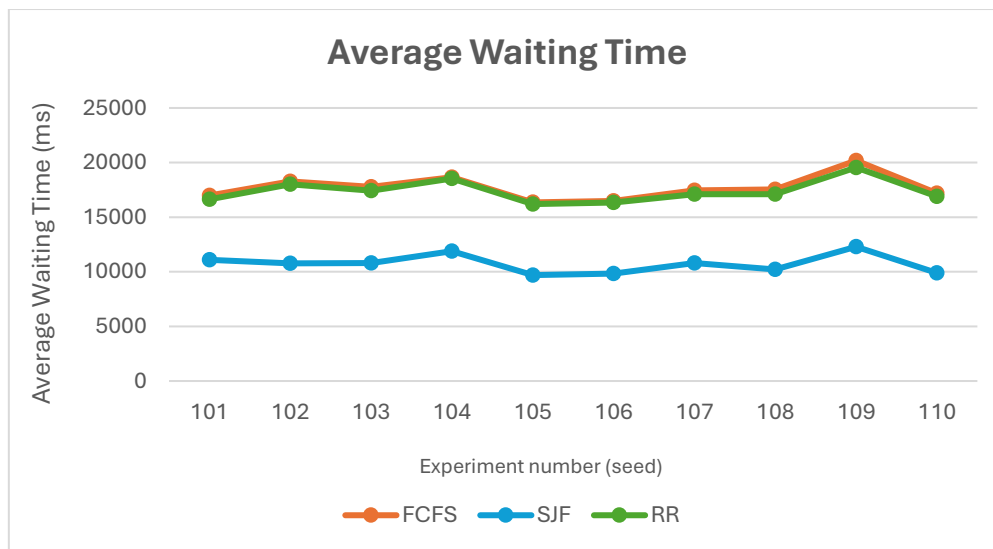
5.2 Waiting Time



Scheduling Algorithm	Average Time (ms)	Median Time (ms)	Variance (ms)
FCFS	16980	4904	10694367.01

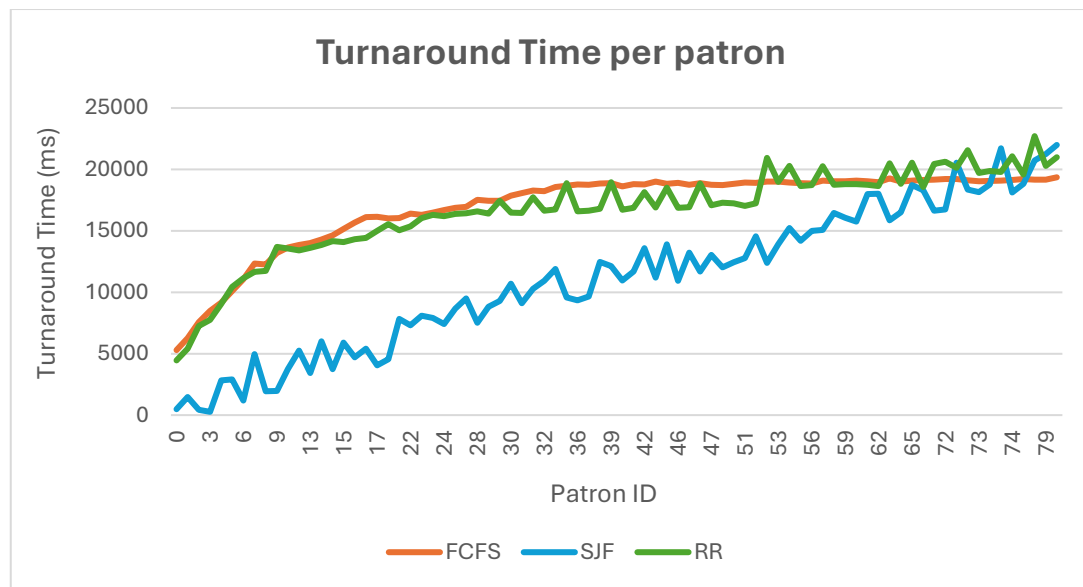
SJF	11087	4845	33608448.23
RR	16638	7634	12905740.34

- SJF (non-pre-emptive) is expected to have the lowest average waiting times which is confirmed by the graph above. This is because it prioritises the drinks with the shortest execution times thus leading to overall faster completion and lower waiting times.
- FCFS and RR have significantly higher waiting times, however towards the end the last few patrons from ID number 62 have similar high waiting times across all algorithms.
- RR has high waiting times due to its frequent context switching.
- SJF has significantly higher waiting time towards the end contributing to its high variance (as seen in variance). It takes very long for long orders making it unfair for jobs that are long but arrived much earlier.



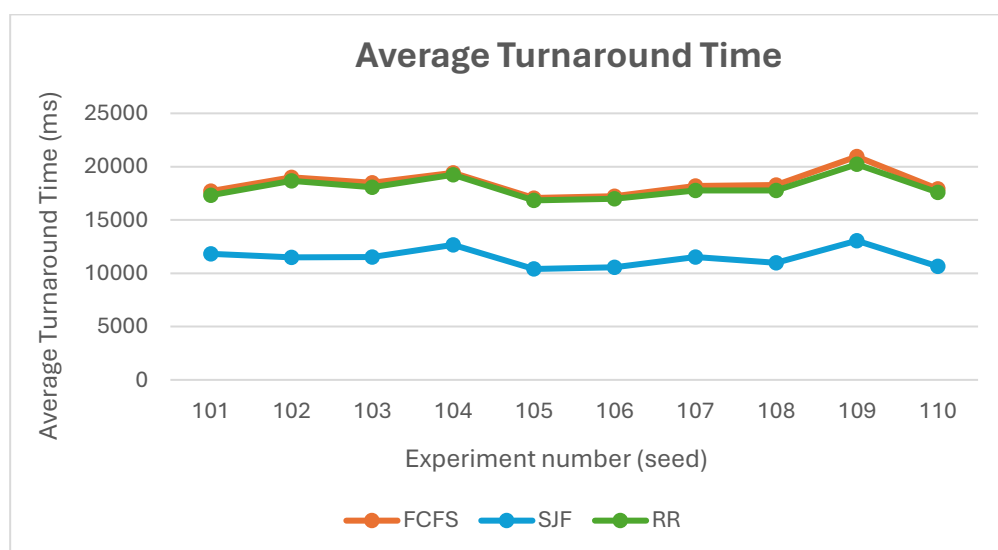
- As seen in experiment 1 the values keep changing across the different experiments however there are not drastic changes thus confirming the stability of the experiments.
- Overall while SJF has lowest waiting time it has a very high variance making it unsuitable, RR shows a moderate waiting time (lower than FCFS) with comparatively lower variance making it the most suited for best waiting times.

5.3 Turnaround Time



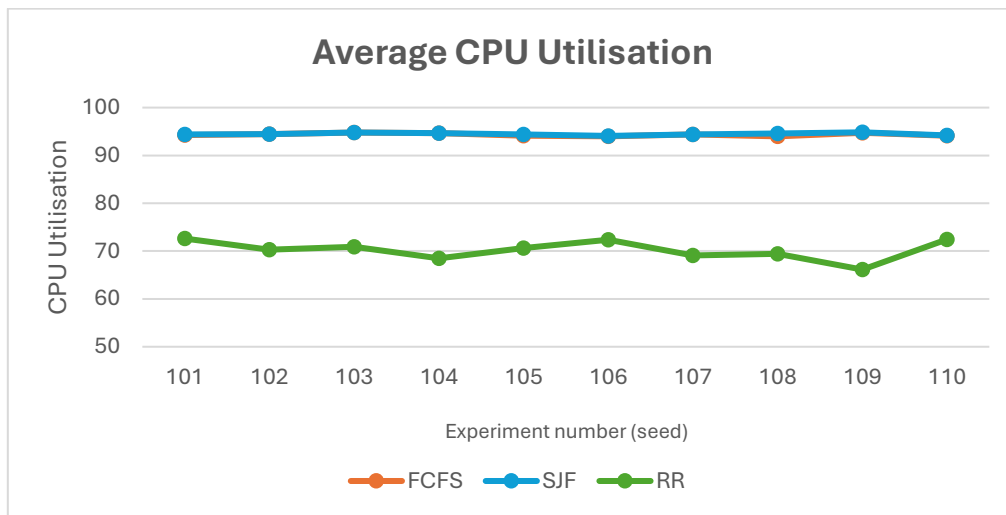
Scheduling Algorithm	Average Time (ms)	Median Time (ms)	Variance (ms)
FCFS	17711	18862	10741169.69
SJF	11818	11814	34556521.4
RR	17307	17366	12736056.8

- As expected, SJF gives the shortest turnaround time per patron as it minimises waiting time and prioritises jobs with shorter execution time thus providing an overall lower turnaround time as can also be seen in the graph.
- FCFS and RR similar to the waiting time analysis, have significantly higher turnaround times for the patrons that arrive much later. SJF follows a similar pattern as its significantly increases turnaround times with values similar to RR and FCFS for patrons with ID high than approximately 62 – resulting in its high variance.
- FCFS suffers due to early long jobs delaying the completion of subsequent ones.



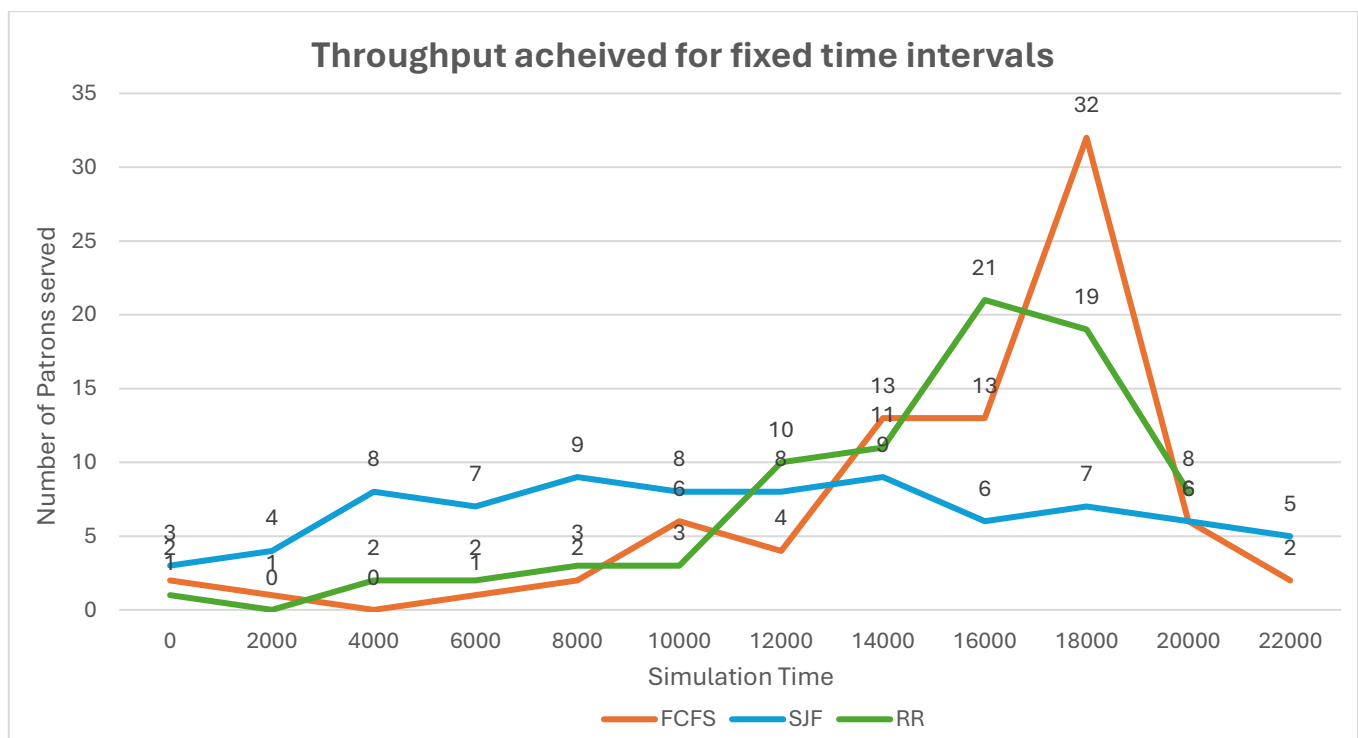
- Overall, across the experiments SJF has the lowest average turnaround time but the highest variance making it unsuitable. RR offers the most balance and fairness between the patrons managing trade-offs between prioritisation and variance.

5.4 CPU Utilisation



- CPU utilisation is expected to be the highest when the CPU spends idle time or context switching time, therefore RR is expected to have the lowest CPU utilisation as is also evident from the results of our experimentation. SJF and FCFS provide the highest CPU utilisation.
- As can be seen from the graph the CPU utilisation is between 65% to 74% for RR while it is much higher around 95% for FCFS and SJF.
- The CPU utilisation remains relatively stable for FCFS and SJF while it fluctuates quite a bit for RR.
- RR has the lowest CPU Utilisation due to its pre-emptive nature and context switch time.

5.5 Throughput

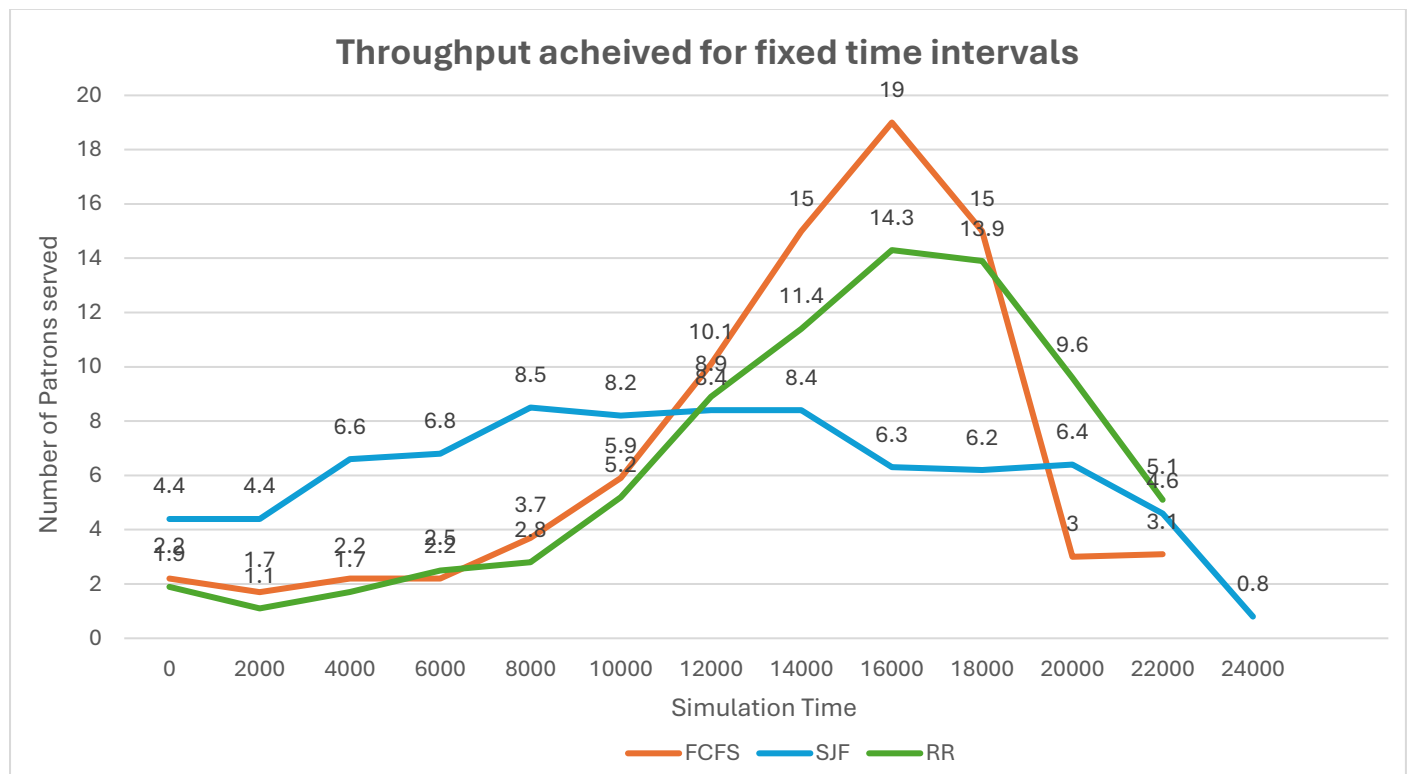


The graph above is for one simulation of seed = 103 across all algorithms based on Experiment 1.

- The throughput is the highest for SJF for about 50% of the simulation time. It also remains the most stable throughout the simulation.

- The throughput for FCFS and RR is lower in the beginning but increases after half time of the simulation as more process are completed entirely especially patrons whose drinks were being sent to the back of the order queue.

This graph takes an average of throughput value when the experiments are run multiple times.



- Similar results as discussed for the graph from experiment 1 can be seen in this graph with experiment 2.

5.6 Fairness of algorithms

Fairness is the concept of equal CPU access to all processes (patrons). RR is generally known for being the fairest as it gives each patron a turn every quantum cycle. This can be seen in the response time and waiting time which is which the graph is a bit jagged. FCFS can also be considered fair as it prioritises the patrons that arrive earlier however is very unfair to patrons that arrive later resulting in high response times for patrons with high patron IDs. SJF is the least fair has it has very high variances as it prioritises jobs with shorter length and takes very long for jobs with higher execution time. In the context of the barman customers might be happier if they are served based in a RR queue, while customers arriving earlier would be much happier with FCFS to maintain fair service to all customers RR is the best algorithm.

5.7 Predictability of algorithms

Predictability refers to consistent behaviour across different runs which can be analysed using the variance in the results of the average runs. The table below shows variance values of the different metrics for each of the algorithms. The higher the variance the lower the predictability. The data is obtained from the results of experiment 1. The maximum(orange) and minimum(green) values are highlighted in the table.

Scheduler	Turnaround Time	Waiting Time	Response Time
FCFS	10741169.69	10694367.01	2043312.65
RR	12736056.8	12905740.34	4222792.534
SJF	34556521.4	33608448.23	18668360.12

- It is evident that FCFS is the most predictable algorithm for all metrics as seen above while SJF is the least predictable algorithm. These observations are based on the variance.

The variance obtained in the average values across the different seeds which show how stable the algorithms are for different runs is obtained as seen in the table below. This is based on experiment 2.

Scheduler	Turnaround Time	Waiting Time	Response Time	CPU Utilisation
FCFS	1312863.98	1279890	26540.13	0.088366
RR	1113823.42	1101115	71475.95	4.16704
SJF	776929.884	751835.2	703219.7	0.058654

- FCFS varies a lot across the different runs for turnaround time and waiting time. While SJF is varies much less for these metrics.
- SJF has a high variance for response time across the different runs while FCFS has the lowest variance.

Taking into consideration results of both tables it can be seen that while SJF is relatively stable across the different runs, it varies the most and is the least predictable for each run making it unsuitable for the barman scenario as patrons do not want to be treated so differently. Also SJF is hard to predict generally as these results are based of knowing future orders while in reality that is impossible. FCFS is the best algorithm when it comes to predictability.

5.8 Starvation considerations

RR is optimised to prevent starvation while SJF is prone to it as it keeps processes with longer execution times waiting for very long. As can be seen it has significantly high turnaround, waiting and response times towards patrons arriving much later and with drinks with longer execution time – which can possibly result in starvation of these processes. RR is the safest against starvation as it ensures regular time slices to the CPU. FCFS does not generally suffer from starvation as it serves all process in order unless the system is extremely overloaded.

6. Conclusion

6.1 Best Algorithm Recommendation

In the context of this experiment while SJF gave the shortest times it remains highly variant as discussed in more detail under the sections for each of the metrics. It is also prone to starvation, is highly unpredictable and unfair to drinks with longer waiting time. FCFS provides lower variance and relatively good average timings with high predictability but can reduce efficiency. As can be seen in the graph for throughput from experiment 1 it results in the longest simulation times. It is not the most efficient algorithm as it does not optimise the drinks process and is unfair to patrons who arrive much later as they might have to wait for the whole queue before they can be served a drink – even in cases where giving them a drink might result

in overall efficiency increase through shorter simulation times. RR performs very similarly to FCFS with its waiting time, turnaround times and response time. While its only drawback is that it not the most predictable it still can be considered fairly predictable as its value of variance are not that much higher than FCFS. It can be considered very fair as all patrons are given a chance to have a drink by distributing preparation time across the time quantum. In a real-life barman scenario, especially in the case of a crowded bar the overall customers would be considered more satisfied if they got a drink before waiting in a very long queue. RR allows all customers to start having a drink as can be seen from the graph a lot of customers have one of the lowest response times. This will leave more customers satisfied therefore RR can be concluded as the best algorithm.