Operating Systems Report

Candidate Number: 246529

Table of Contents

1 Experiment 1

1.1 Introduction

In this experiment I am going to be evaluating how the number of processes scheduled will affect the average waiting time of the different CPU algorithms to determine what implementation of each algorithm is the most optimal, if at all.

For the following algorithms, increasing the amount of processes delivered should not diminish the overall performance of the algorithm, and I expect to see similar results in the efficiency of each algorithm, with the average waiting time of each algorithm staying roughly the same.

Hypothesis:
Increasing the number of processes given as an input will not increase the average waiting time of each scheduler.

1.2 Methodology

The structure of this experiment is as follows:

```
experiment1
  ↳seed$K
    ↳input_parameters.prp
     input.in
     output$SCHEDULER.out
     simulator_parameters$SCHEDULER.prp
```

Where:

- $K: 1, 2, 3
- $SCHEDULER: FCFS, FeedbackRR, IdealSJF, RR, SJF

Independent variable: `numberOfProcesses`
Dependent variable: `waitingTime`

Generating a wider variety of results is an important consideration so I decided to include 3 different seeds in each experiment. By doing this, I can ensure that a wider range of possible outcomes is explored, as each seed can guarantee a truly unique set of inputs in my input.in file, and therefore a unique set of outputs that I can measure. The seeds I have used are randomly generated 15 digit numbers to ensure the outcomes are not predetermined in any way.

For seed 1, I changed the number of processes to 10, for seed 2 I changed the number of processes to 20, and for seed 3 I changed the number of processes to 30. This approach allowed me to obtain a broad understanding of how each CPU performs under different workloads.

Input Parameters:

```
numberOfProcesses=10
staticPriority=0
meanInterArrival=150.
0
meanCpuBurst=15.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=638720198452787
```

```
numberOfProcesses=20
staticPriority=0
meanInterArrival=150.
0
meanCpuBurst=15.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=717776928100739
```

```
numberOfProcesses=30
staticPriority=0
meanInterArrival=150.
0
meanCpuBurst=15.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=896394428968565
```

Simulator Parameters:

```
scheduler=$Scheduler
timeLimit=10000
periodic=false
interruptTime=0
timeQuantum=10
initialBurstEstimate=10
alphaBurstEstimate=0.5
```

Where:

- $: Fcfs, FeedbackRR, IdealSJF, RR, SJF

*The other variables in the simulator parameters will remain the same for each scheduler

1.3 Results

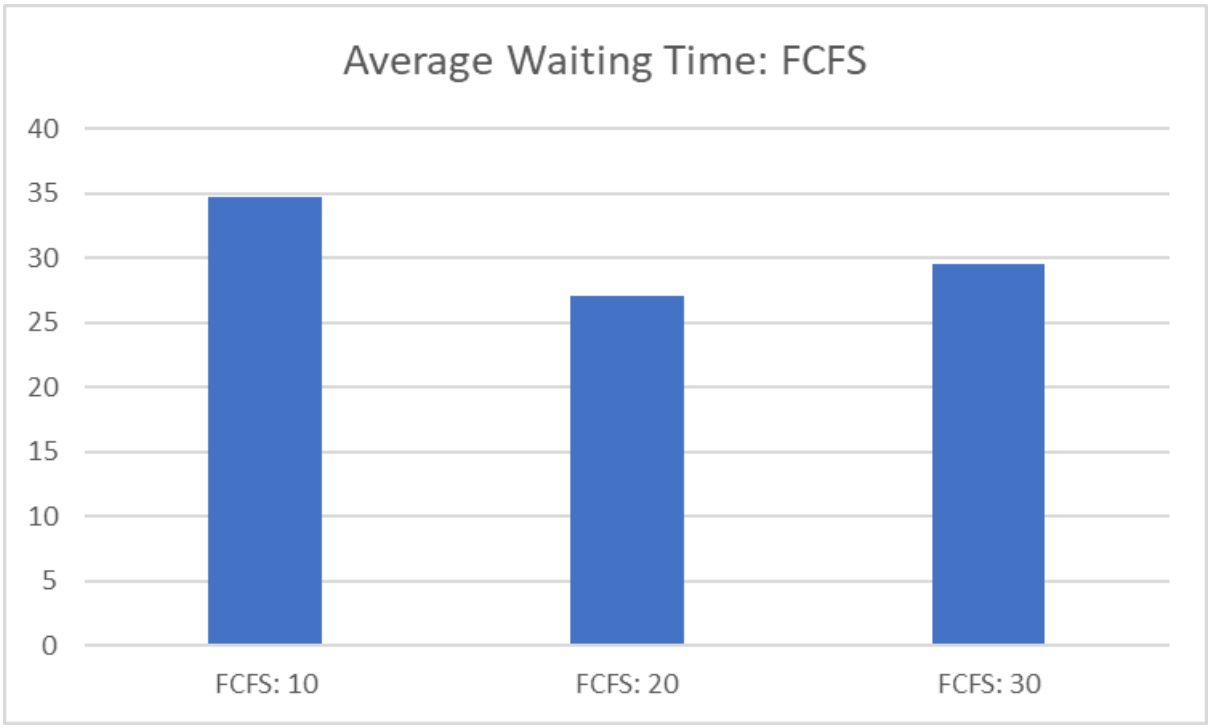

Figure 1.a

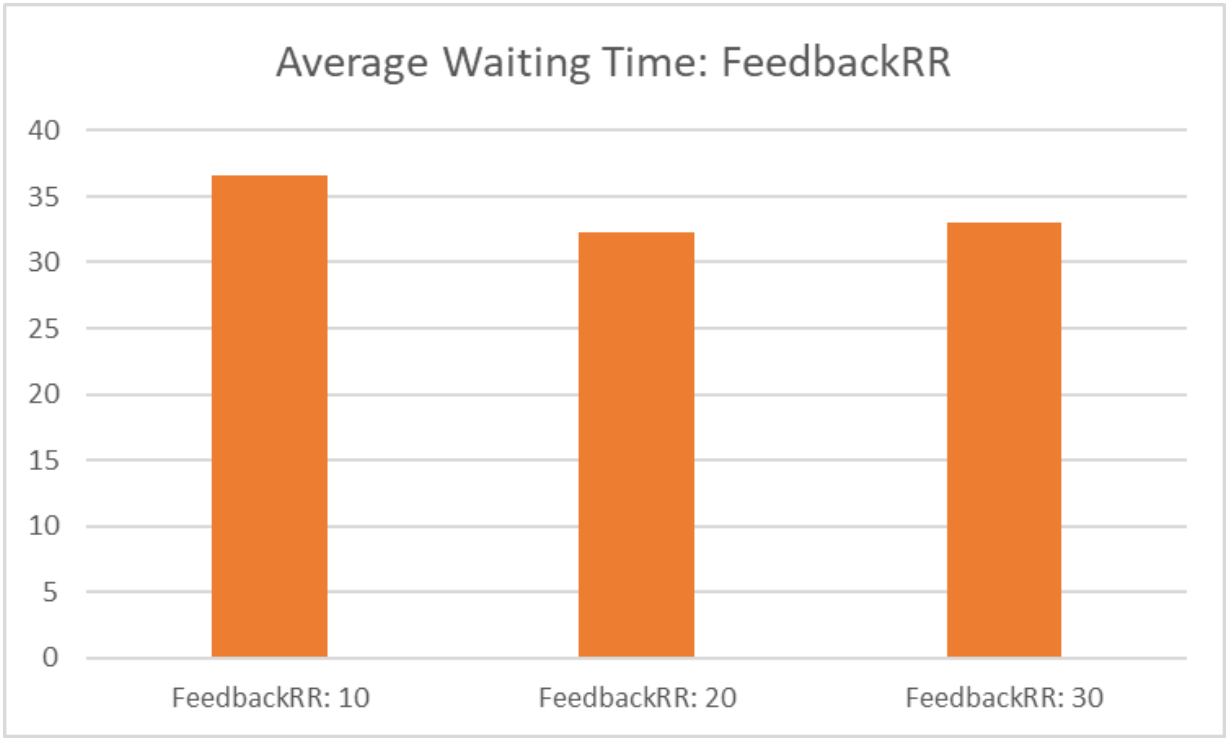

Figure 1.b

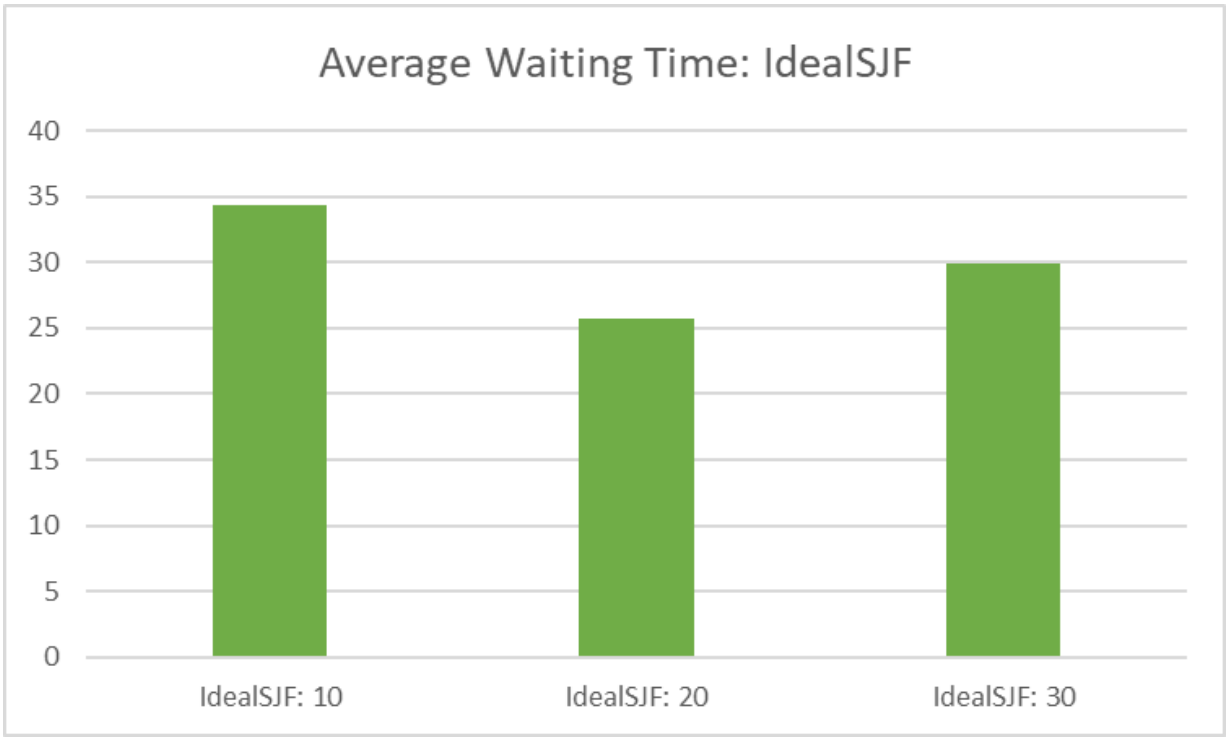

Figure 1.c

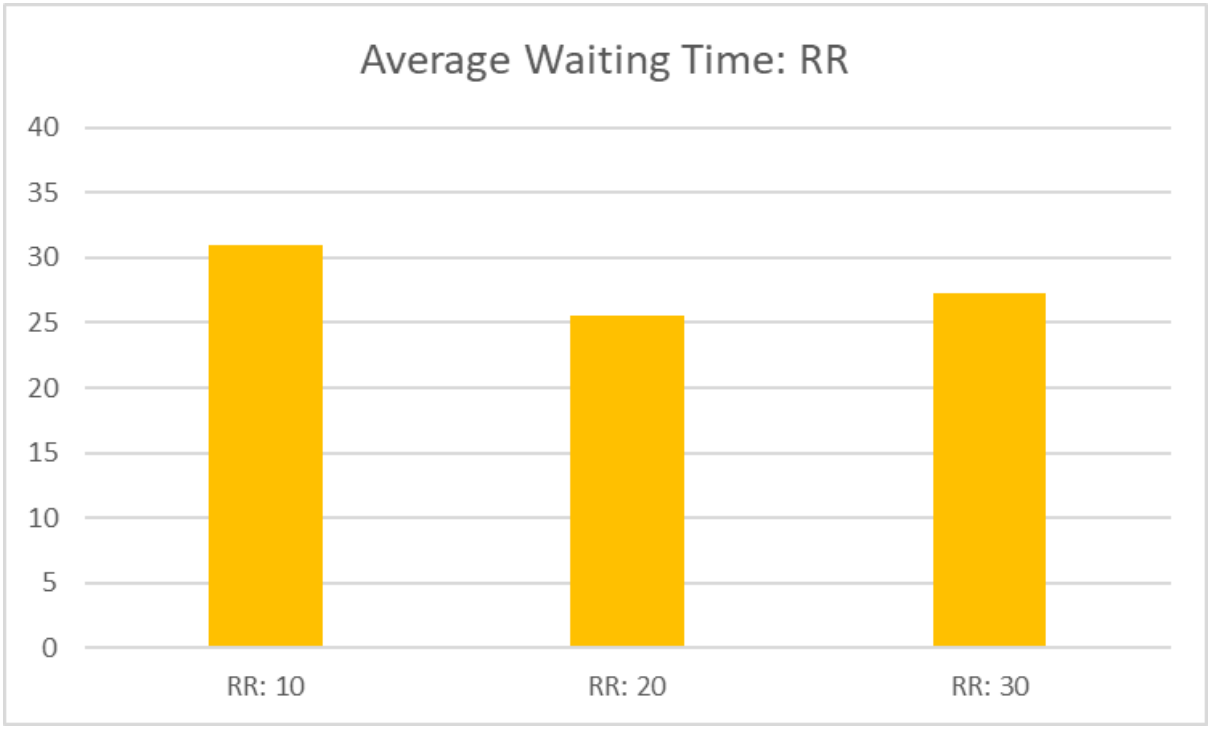

Figure 1.d

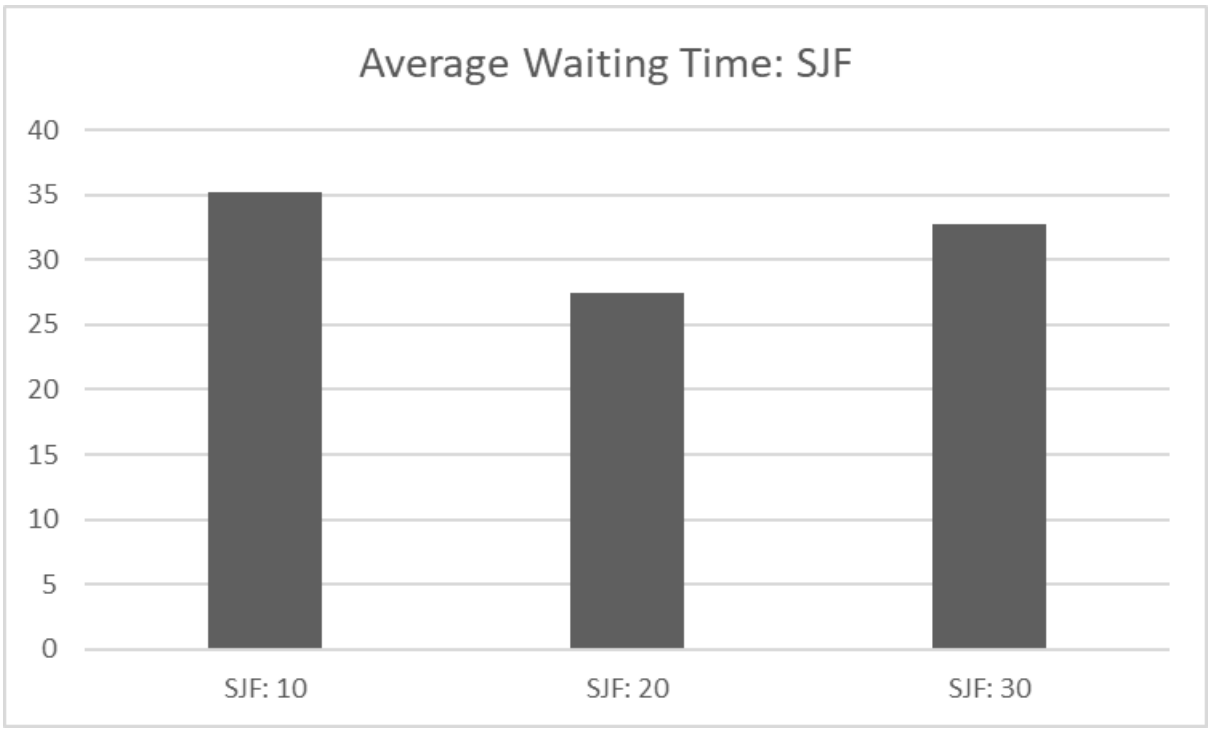

Figure 1.e

For all my graphs, the X-Axis contains the name of the scheduler along with the number of processes executed i.e. 10, 20, 30, in the form of a bar chart to help visualise the results. The Y-Axis is numbered 0-40 as all of the data for the schedulers were between these values.

To calculate the Average Waiting Time, I took the sum of the `waitingTime` column and then divided it by the number of processes respectively.

1.4 Discussion

Looking at the graphs created in the 1.3 Results section, every CPU seems to support my original claim, as in all the graphs, the result with the least amount of processes (10) has the highest average waiting time, and the other two columns never go higher. This could be due to a variety of reasons for example; the scheduling algorithms are generally inefficient at dealing with a low workload and instead can distribute its resources better with a high workload, or the arrival rate of each process is higher in the schedulers with a lower number of overall processes, meaning there is more congestion leading to a higher overall waiting time.

Furthermore, all the graphs follow a similar pattern of being higher for the lowest number of processes, and lowest for the middle number of processes, with the CPU's with the highest number of processes laying somewhere in between. While some of the schedulers had a more similar average waiting time, I couldn't quite understand why the middle number of processes performed more efficiently than the others but had a few guesses. One possible explanation could be that the workload was better balanced i.e. the input.in file in the scheduler with 20 processes produced an easier workload, but only by running more experiments and gaining a greater number of results would I have maybe seen more linear looking graphs.

1.5 Conclusion

Original Hypothesis: "Increasing the number of processes given as an input will not increase the average waiting time of each scheduler."

In conclusion, my initial hypothesis is supported by the investigation and subsequent data produced. The results show that there is no major variation in the average waiting time of each scheduler when the number of processes is increased. However, it is important to understand that other factors such as the different input parameters could still affect the overall efficiency of the scheduling algorithms, and that by running more tests on this same experiment, the validity of my results would only increase.

2 Experiment 2

2.1 Introduction

In this experiment I am going to be investigating how increasing the mean CPU burst time will affect the different scheduling algorithms and determine which one will perform best.

Increasing the mean CPU burst time will have a varying effect on the different scheduling algorithms with some performing better than others. For example, SJF schedulers execute processes based on the next estimated CPU burst time, with the shortest job being scheduled first. This means they perform well when the mean CPU burst time is high because it allows shorter processes to complete quickly and reduces the waiting time for longer processes. Furthermore, FeedbackRR also assigns priority to shorter processes so may outperform the other schedulers. On the other hand, RR is designed to provide better responsiveness to diverse workloads, but they may not perform as well when the mean CPU burst time is high, leading to an increased turnaround time. As for FCFS, this algorithm executes processes fairly without considering the length of the CPU burst, and so increasing the mean CPU burst will see them perform worse than the other scheduling algorithms.

Hypothesis:
The algorithms that schedule processes based on priority will be better at dealing with an increased mean CPU burst time than the others.

2.2 Methodology

The structure of this experiment is as follows:

```
experiment2
  ↳seed$K
    ↳input_parameters.prp
     input.in
     output$SCHEDULER$K.out
     simulator_parameters$SCHEDULER.prp
```

Where:

- $K: 1, 2, 3, 4, 5
- $SCHEDULER: FCFS, FeedbackRR, IdealSJF, RR, SJF

Independent variable: `meanCpuBurst`
Dependent variable: `turnaroundTime`

By changing the mean CPU burst it will allow me to evaluate the performance of different scheduling algorithms in different scenarios, so for future reference I can use the most optimal CPU for a high mean burst time.

For this experiment, I have decided to increase the number of changes to my independent variable. Learning from my previous experiment, it was quite hard to gauge the scheduler's performance with only 3 seeds, so I am now going to use 5. This will allow me to draw more robust conclusions and develop my understanding and will help to avoid anomalies.
I will be increasing the mean CPU burst time to 75 as I believe this is a large enough increase to simulate a more intensive workload which will really highlight differences in performance. This will allow me to observe a wide variety of data and assess the overall performances of the schedulers fairly.

Input Parameters:

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=75.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=198950578717713
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=75.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=770207193099024
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=75.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=365022231390681
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=75.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=947591456890826
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=75.0
meanIOBurst=15.0
meanNumberBursts=2.0
seed=259796970117300
```

Simulator Parameters:

```
scheduler=$Scheduler
timeLimit=10000
periodic=false
interruptTime=0
timeQuantum=10
initialBurstEstimate=10
alphaBurstEstimate=0.5
```

Where:

- $: Fcfs, FeedbackRR, IdealSJF, RR, SJF

*The other variables in the simulator parameters will remain the same for each scheduler

2.3 Results

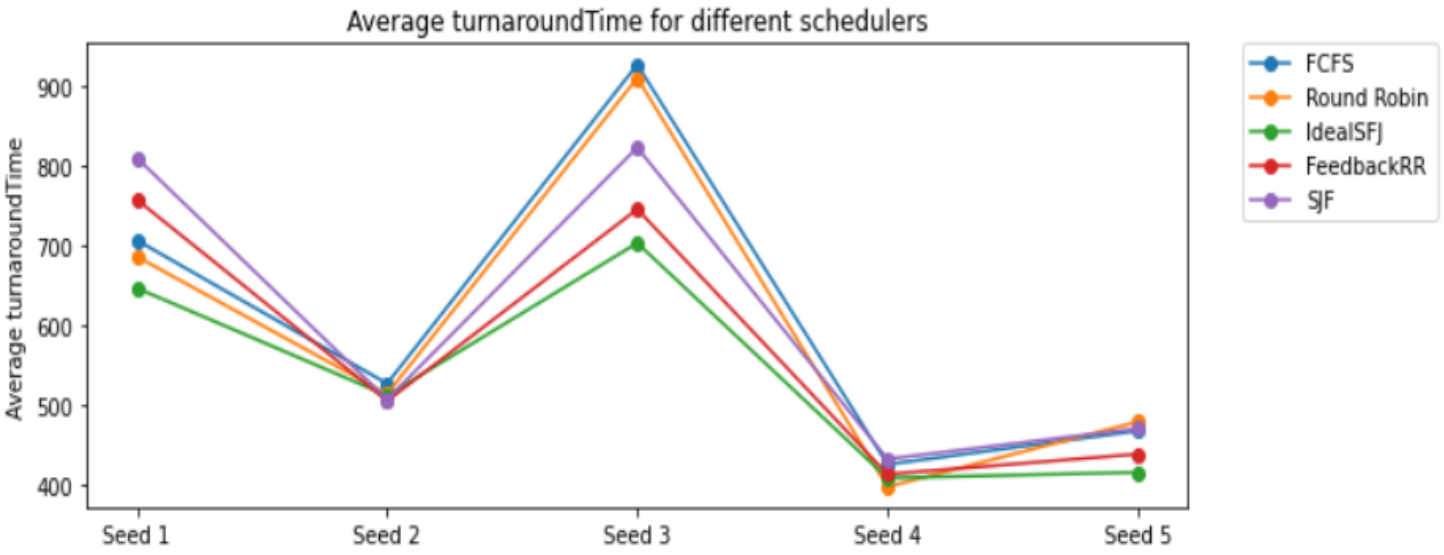

Figure 2.a

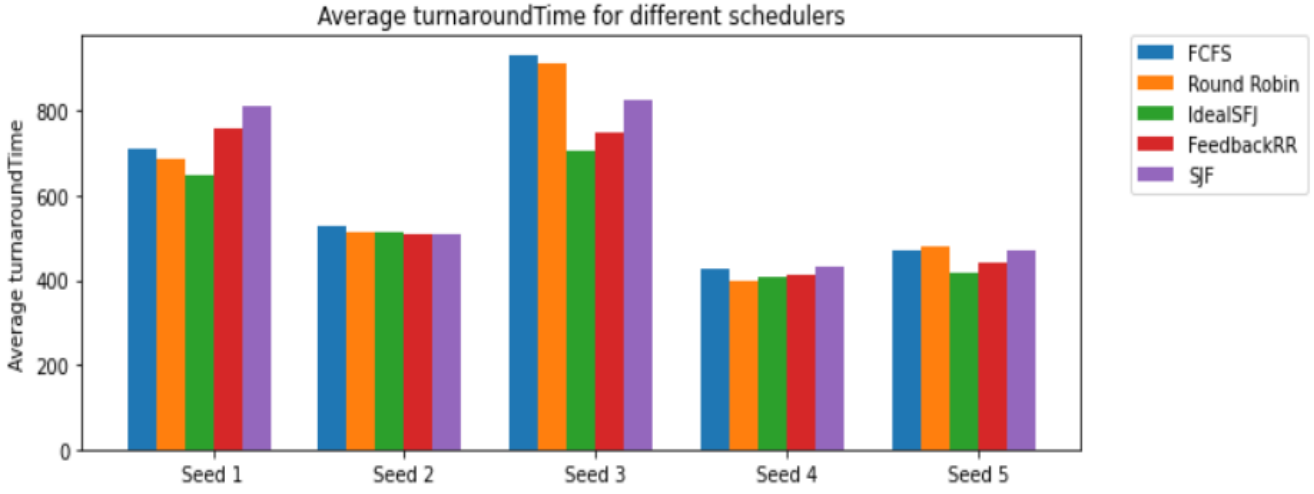

Figure 2.b

2.4 Discussion

In this experiment, I decided to include all the different schedulers in the same graphs to better visualise the differences in performance between them. As you can see from Figure 2.b, the general workload for seeds 1 and 3 are much higher than the others. This also seemed to produce a larger discrepancy between each of the schedulers which could be due to the fact that with a higher average turnaround time it is less likely the algorithms will be able to prioritise shorter processes over longer ones.

As predicted in my hypothesis, the priority based schedulers did perform better on average than the other ones, most notably the IdealSJF scheduler consistently produced the lowest average turnaround time for each seed which could be due to the fact it knows the next burst time of the processes and prioritised the processes in the best order. Furthermore, FeedbackRR was also very consistent at producing a low average turnaround time as processes waiting can receive feedback that allows processes to move between priority queues based on their behaviour such as long running processes remaining on the CPU for a large amount of time.

Finally, FCFS performed the worst in almost every run which could be due to shorter processes not being able to access the CPU as it may be occupied by a longer running process leading to a backlog of processes and longer waiting times.

2.5 Conclusion

Original Hypothesis: “The algorithms that schedule processes based on priority will be better at dealing with an increased mean CPU burst time than the others.”

In conclusion, my initial hypothesis is supported by the investigation and subsequent data produced. The results highlight that more advanced schedulers such as SJF and FeedbackRR perform better when dealing with an increased mean CPU burst time which is a significant factor when considering CPU performance.

3 Experiment 3

3.1 Introduction

In this experiment I am going to be investigating how changing the mean IO burst will affect the different scheduling algorithms and determine which one will perform best.

The IO burst is the amount of time a process waits for input-output before needing CPU time, and the duration of these bursts can significantly impact the overall performance of the system. A CPU burst is the amount of time a process uses the CPU until it starts waiting for some input or is interrupted by some other process. An I/O-bound program typically has many short CPU bursts so it will be interesting to see what effect a high I/O burst can have on a relatively low mean CPU burst time in comparison. Like experiment 2, the FCFS and RR scheduling algorithms have no capability in prioritising different processes which can cause other processes to wait longer before getting access to the CPU, decreasing the overall performance. The schedulers with more advanced algorithms are therefore expected to outperform the other two and minimise wait time between I/O bursts.

Hypothesis:
The more advanced schedulers are more optimised at dealing with workloads with increased I/O bursts.

3.2 Methodology

The structure of this experiment is as follows:

```
experiment3
  ↳seed$K
     ↳input_parameters.prp
      input.in
      output$SCHEDULER$K.out
      simulator_parameters$SCHEDULER.prp
```

Where:

- $K: 1, 2, 3, 4, 5
- $SCHEDULER: FCFS, FeedbackRR, IdealSJF, RR, SJF

Independent variable: `meanIOBurst`
Dependent variable: `turnaroundTime, waitingTime, responseTime`

Following the logic with my previous experiment, I will be again using 5 different seeds to produce more varied data and avoid anomalies.

For this experiment, I will be assessing the turnaround time, waiting time and the response time of all 5 schedulers. This will allow me to more accurately assess the robustness and accuracy of my results because some schedulers may be better than others in turnaround time,

where their response time may fall, or are more efficient with that specific number of processes. I will be keeping the number of processes and mean CPU burst at 25 and 15.0 respectively to allow the change in I/O to properly show over a controlled set of inputs, and will be increasing the mean I/O burst to 30.0 as to differ from the previous experiment where the expected workload was very high, this should be more balanced to make the experiment fairer. This will allow me to examine a wide range of data and carefully assess the scheduler's overall performance.

Input Parameters:

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=15.0
meanIOBurst=30.0
meanNumberBursts=2.0
seed=636713454435660
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=15.0
meanIOBurst=30.0
meanNumberBursts=2.0
seed=168121261590752
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=15.0
meanIOBurst=30.0
meanNumberBursts=2.0
seed=904514056194773
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=15.0
meanIOBurst=30.0
meanNumberBursts=2.0
seed=518734418753605
```

```
numberOfProcesses=25
staticPriority=0
meanInterArrival=150.0
meanCpuBurst=15.0
meanIOBurst=30.0
meanNumberBursts=2.0
seed=459490129405825
```

Simulator Parameters:

```
scheduler=$Scheduler
timeLimit=10000
periodic=false
interruptTime=0
timeQuantum=10
initialBurstEstimate=10
alphaBurstEstimate=0.5
```

Where:

- $: Fcfs, FeedbackRR, IdealSJF, RR, SJF

*The other variables in the simulator parameters will remain the same for each scheduler

3.3 Results

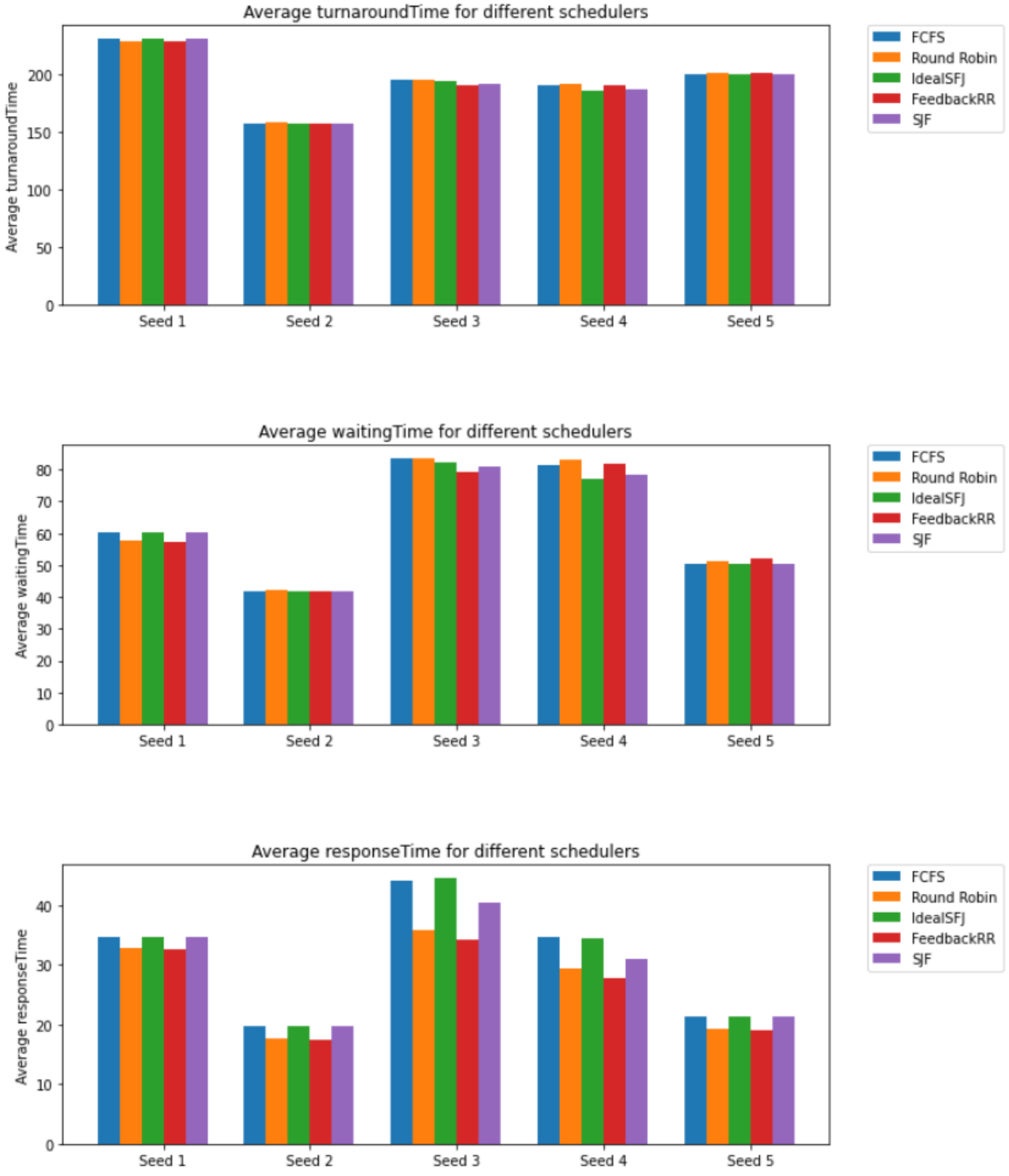

Figure 3.a

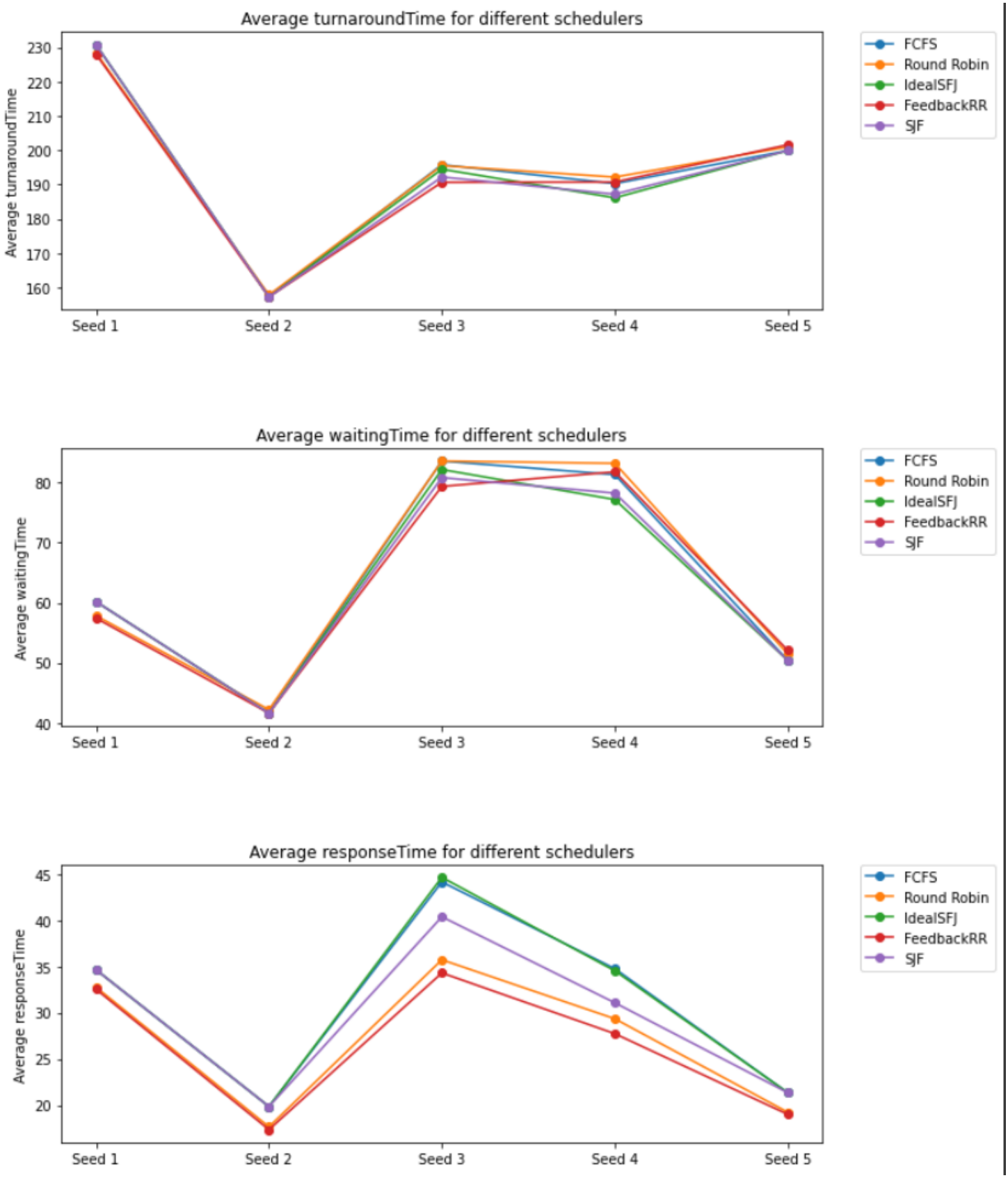
Average turnaroundTime for different schedulers
Average turnaroundTime
230
220
210
200
190
180
170
160
Seed 1
Seed 2
Seed 3
Seed 4
Seed 5
FCFS
Round Robin
IdealSFJ
FeedbackRR
SJF
Average waitingTime for different schedulers
Average waitingTime
80
70
60
50
40
Seed 1
Seed 2
Seed 3
Seed 4
Seed 5
FCFS
Round Robin
IdealSFJ
FeedbackRR
SJF
Average responseTime for different schedulers
Average responseTime
45
40
35
30
25
20
Seed 1
Seed 2
Seed 3
Seed 4
Seed 5
FCFS
Round Robin
IdealSFJ
FeedbackRR
SJF

Figure 2.b

3.4 Discussion

As previously mentioned, In this experiment, I decided to include three different metrics to measure the performance of the different scheduling algorithms. In all three tables, the average wait time, turnaround time and response time all decrease going from seed 1 to seed 2, before increasing for the other seeds. This generally indicates that seed 2 had a much less diverse workload as all of the scheduling algorithms performed very highly on this seed in particular.

I brought up the point earlier that some schedulers may perform better than others in one performance metric and worse in others, and from the graphs in Figure 3.a we can clearly see that both the scheduling algorithms that use Round Robin perform much better at response time than waiting or turnaround time. This could be because these algorithms are preemptive which means that processes are allocated a certain amount of time on the CPU before they are removed and added back to the ready queue, so all processes get a fair amount of time on the CPU improving responsiveness.

As for the average turnaround and waiting time, we can clearly see that again the best performing algorithms are IdealSJF, SJF and FeedbackRR. This is because of the advanced nature of these algorithms, allowing priority based scheduling which produces a more efficient CPU, which directly aligns with my hypothesis.

3.5 Conclusion

Original Hypothesis: “The more advanced schedulers are more optimised at dealing with workloads with increased I/O bursts.”

In summary, after evaluating the results data along with the characteristics of the scheduling algorithms it is plausible to agree with the above hypothesis. These schedulers have been developed with more efficient algorithms and scheduling techniques, allowing them to allocate resources more efficiently, prioritise I/O-bound processes, and minimise waiting time. However it is important to note that other factors may not threaten the validity of this experiment. Factors such as parameter variability where variables like time quantum have not been altered may hinder the performance of the Round Robin based schedulers, where only an increased number of experiments could give fairer results.

4 Validation

4.1 Validation of performance metrics

Average Waiting Time: Waiting time is the total time spent by the process in the ready state waiting for the CPU. Different CPU scheduling algorithms have different strategies for selecting which process to execute next from the ready queue such as Round Robin allocating a specific time slice to each process or SJF scheduling processes based on priority. By comparing the average waiting times of different scheduling algorithms, we can determine the algorithms with lower wait times are most optimal and have a better overall performance.

Average Turnaround Time: Turnaround time is the total amount of time spent by the process from when it first enters the ready state to its completion. This measurement is important for assessing a system's overall performance because it indicates how long it takes for a process to execute from start to finish. It includes both waiting time and execution time so is arguably a fairer metric to measure when considering the efficiency of the scheduling algorithms as a whole. By comparing the average turnaround times of different schedulers, it helps in identifying the best algorithm for a particular system, optimising its performance.

Average Response Time: Response time is the time spent when the process is in the ready state and gets the CPU for the first time. This metric is important because it helps us assess the responsiveness of a system, which is critical for interactive systems where users expect their requests to be processed quickly and efficiently. By measuring the response time of different CPU scheduling algorithms, we can evaluate their performances and determine which one provides the best user experience.

4.2 Threats to validity

Parameter variability: This could potentially cause a concern as in each of my experiments only one metric was changed. Schedulers such as Round Robin and FeedbackRR rely on a time quantum to execute its processes, and that variable was not changed in my experiments which could have given the other processes an upper hand.

Seed variability: Throughout my experiment I used a system of using a random number generator to generate my seeds. While this often is seen as the fairest way to do so, some schedulers are bound to perform better than others for a specific workload, and with my experiments only using a maximum of 5 seeds, the limited variability can prevent the detection of weaknesses in the algorithm that may only be exposed under certain conditions or with specific types of workloads.

Performance metrics: The metrics I used to measure performance were turnaround time, waiting time and response time. This generally provides a good amount of data when considering CPU performance, however as seen from my results, the same three schedulers seemed to perform better being IdealSJF, SJF and FeedbackRR. This indicates that these scheduling algorithms are a lot more efficient when discussing these metrics but may not be

when considering others such as CPU utilisation. Furthermore, the two SJF algorithms were not as good as the RR schedulers when reading from the average response time, which may be preferred for some systems.