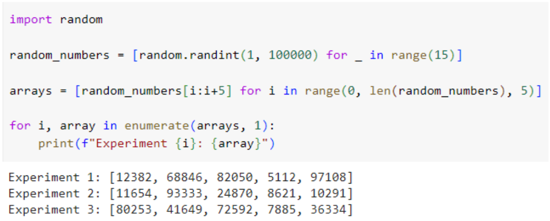
Investigating the differences in CPU schedulers

Operating Systems coursework – 262924 – 19th March 2024

Contents

I. Running the experiments

To run the experiments, you will need to execute the bash script. This script, named run.sh, has saved logs in run.log. Within this bash script, there are 15 different seeds, with 5 for each experiment. These seeds are used in the input generator to reproduce identical input files when all parameters remain the same. These seeds have been randomly generated using a Python script that produces 15 random numbers between 1 and 100,000. Additionally, the bash script defines both the simulator and input parameters for each experiment. These parameters are stored in the input_parameters and simulator_parameters folders within each experiment's folder, identified by their number. Also, these parameters are used to execute the two classes: InputGenerator and Simulator. The resulting output files are saved in input_files and scheduler_outputs, containing 5 schedulers and 5 outputs for each, totalling 25 outputs per experiment.



```
import random

random_numbers = [random.randint(1, 100000) for _ in range(15)]

arrays = [random_numbers[i:i+5] for i in range(0, len(random_numbers), 5)]

for i, array in enumerate(arrays, 1):
    print(f"Experiment {i}: {array}")
```

```
Experiment 1: [12382, 68846, 82050, 5112, 97108]
Experiment 2: [11654, 93333, 24870, 8621, 10291]
Experiment 3: [80253, 41649, 72592, 7885, 36334]
```

Additionally, the script.py file can be run after this file, this computes all the necessary metrics that are needed for each experiment and saves these in a file, with each seed, in the corresponding experiment file name Experiment_(x).txt (where x is the experiment number). These results have then consequently been inputted into a spreadsheet program to display graphs for each experiment.

1. Experiment 1: Investigating the throughput and waiting time of the scheduling algorithms with a high number of processes.

1.1 Introduction

In this experiment, we're looking at the performance of scheduling algorithms with a high workload. Scalability is very important in CPU schedulers and gauges how effectively these algorithms manage resources as the workload increases. The aim is to find out which algorithms excel in resource management with a high number of processes, an important part that all CPU schedulers need to consider.

The focus is on the two key metrics, the two metrics are the throughput and the waiting time. The throughput is the measure of the speed at which tasks are completed. Waiting time is the measure of how long tasks are in the ready queue before execution. By looking at these metrics, I gain insights into the effectiveness of scheduling algorithms under high workloads. This will help determine which algorithm works best under these conditions.

I believe that the algorithm that will be most effective in reducing the average waiting time will be the multi-level feedback queue with round robin, this is due to being able to swap out processes when it has used the full-time slice. Additionally, the use of the priority queue will help in selecting the correct process to run.

Also, the scheduler which will perform best based on the throughput is the SJFScheduler using exponential averaging. I think that this will perform best because it is effective in selecting the shortest process in terms of time to execute, increasing the number of processes it can complete.

1.2 Methodology

In this experiment, I use the same input parameters for each iteration of the experiment, only changing the seed. The most important value is the 'numberOfProcesses' and this value is set to 500 to allow a high number of processes to test on each scheduler. The parameters used for the input generator are below:

numberOfProcesses=500

staticPriority=0

meanInterArrival=10

meanCpuBurst=30

meanIoBurst=20

meanNumberBursts=5

seed=(x)

The parameters for the simulator are below:

scheduler=(y)

timeLimit=40000

interruptTime=1

timeQuantum=12

initialBurstEstimate=30

alphaBurstEstimate=0.5

periodic=false

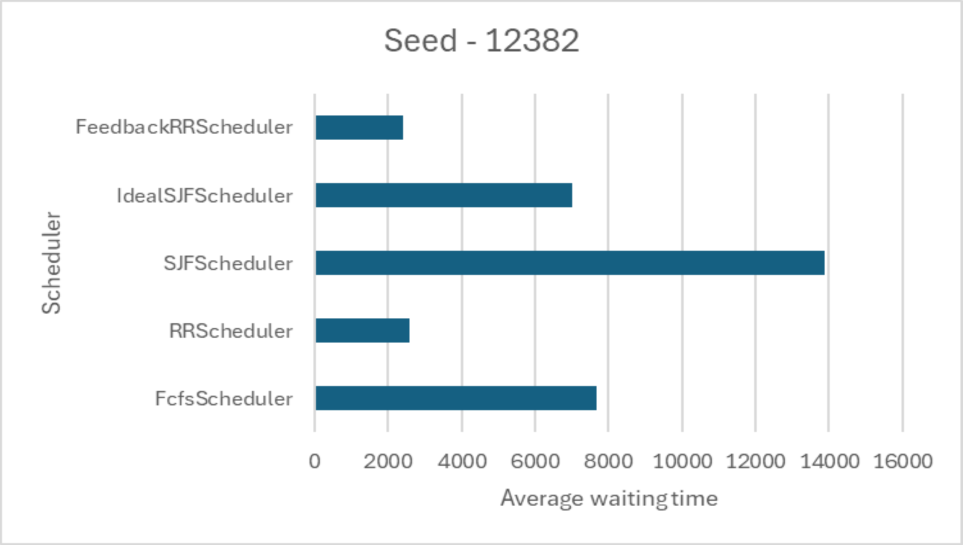
The names of the input files are the input-seed_x where x is the seed used, in this case, the x follows these numbers (12382, 68846, 82050, 5112, 97108). The output files are named output-seed_x following the same number trend. These output files are under a folder with the name of the scheduler the output is from, the y represents the scheduler. All these files can be found under the folder 1.

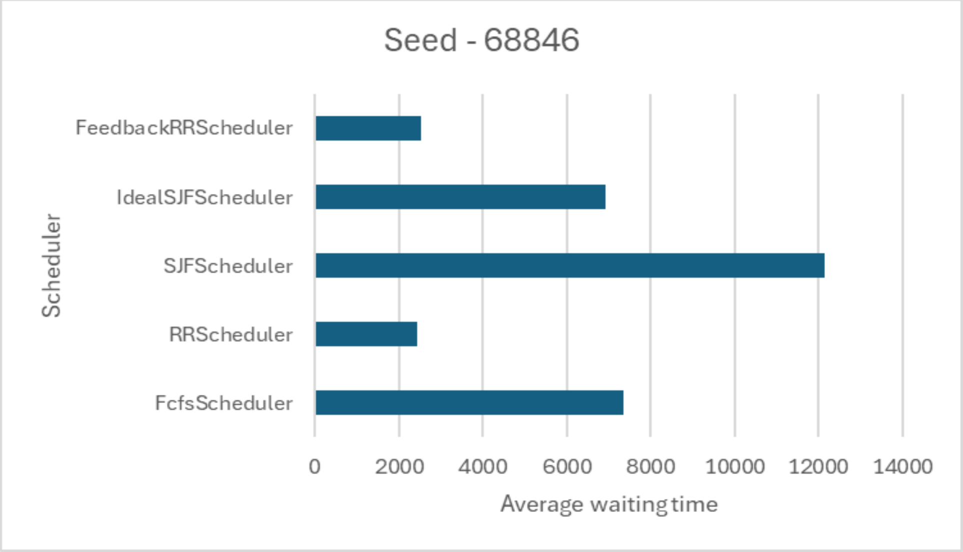
The metrics I have chosen to look at are the throughput (how many processes are completed at each time) and average waiting time (how long processes spend waiting in the ready queue). These are both plotted on two separate graphs. The reason I have chosen these metrics is that these will be a good evaluation of the effectiveness of the scheduler on many processes as the average waiting time will show how well it can swap out processes and not allow processes to stay in the ready queue for a large amount of time. The throughput is good in determining the effectiveness of selecting the correct process to execute.

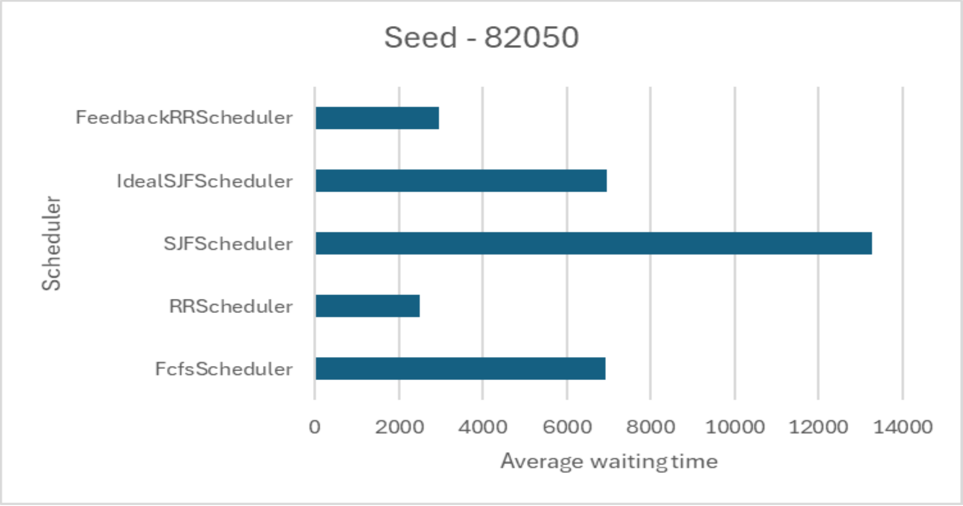
I have chosen a high timelimit to have a high enough time for a lot of these processes to complete and to be able to compute the throughput for the whole experiment. The time quantum of 12 means it has a good balance between not switching between processes too fast but also forcing the round-robin schedule to context switch. I set the initial burst estimate as the mean CPU burst time.

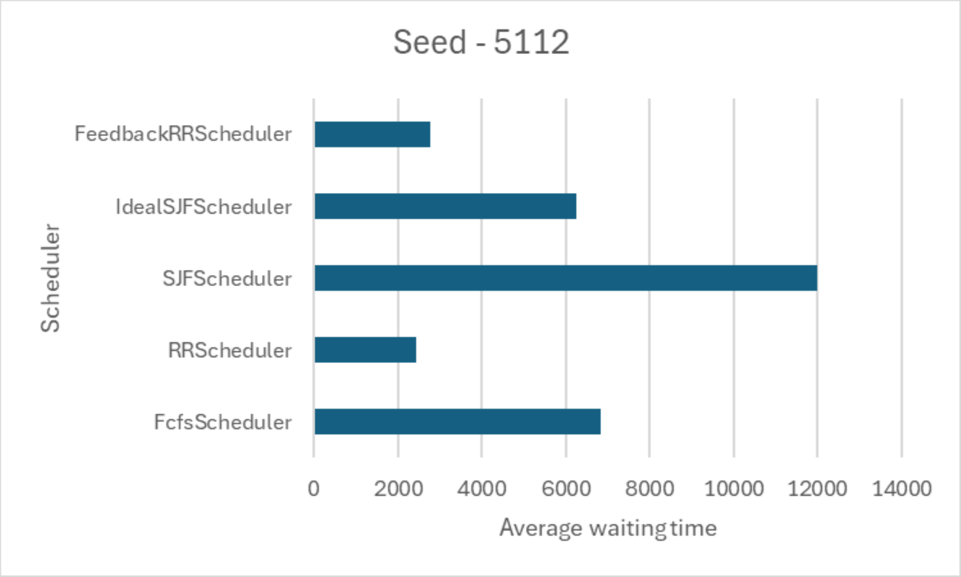
1.3 Results

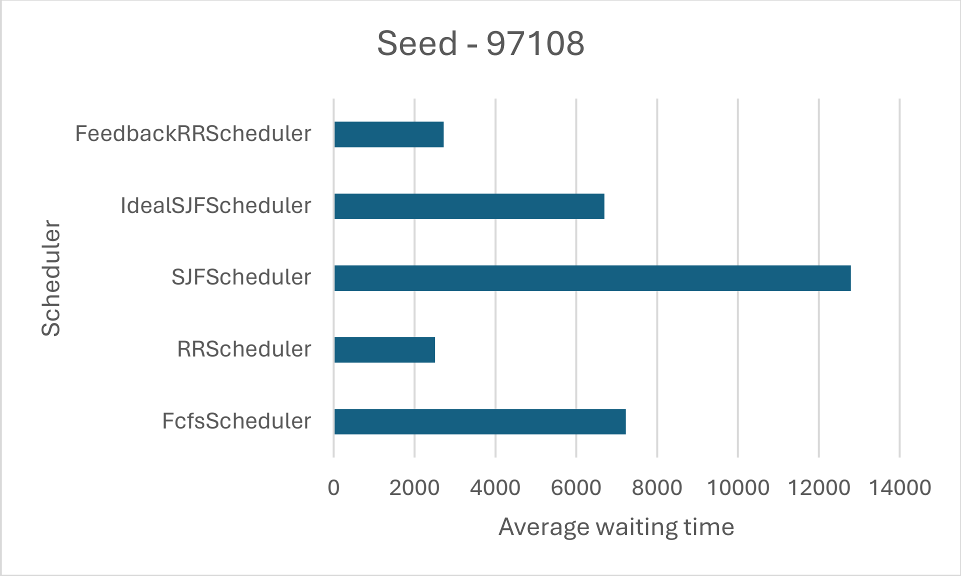
Average waiting times for the 5 seeds



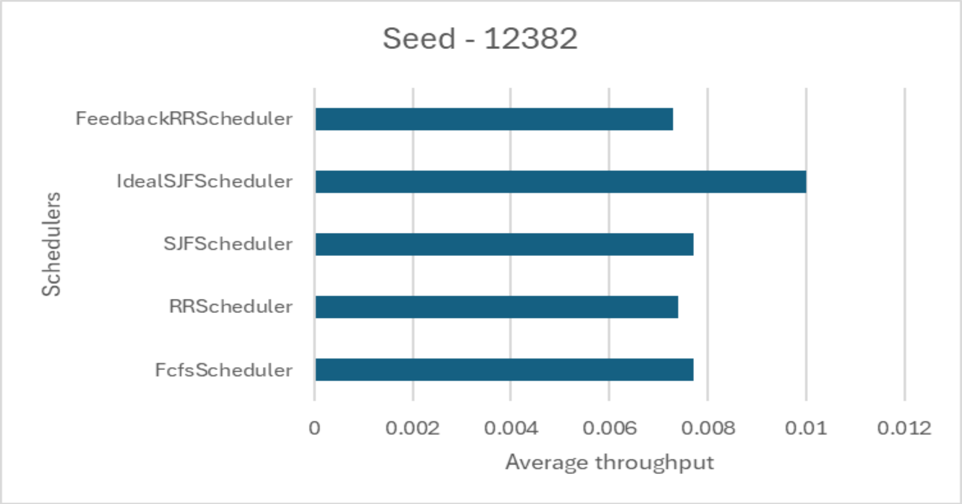


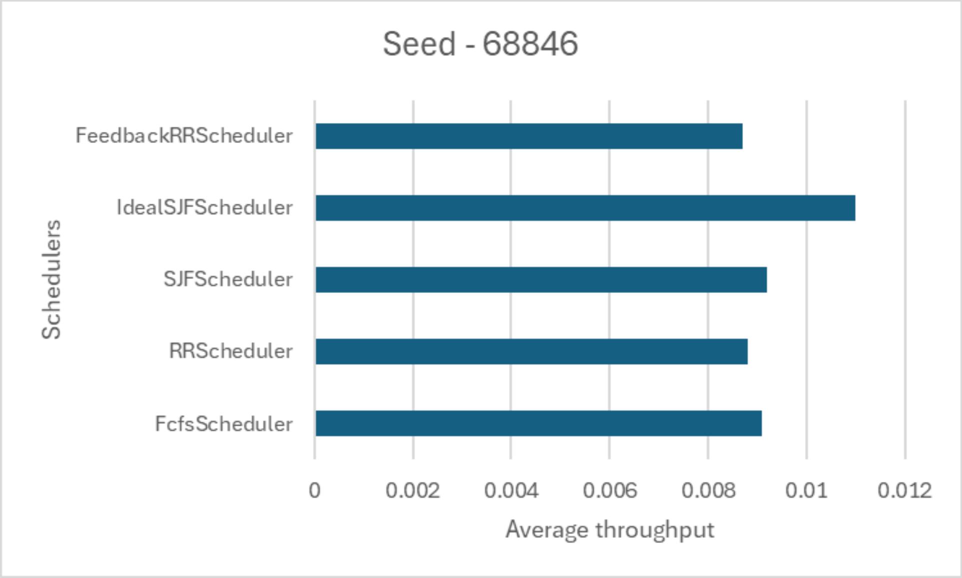


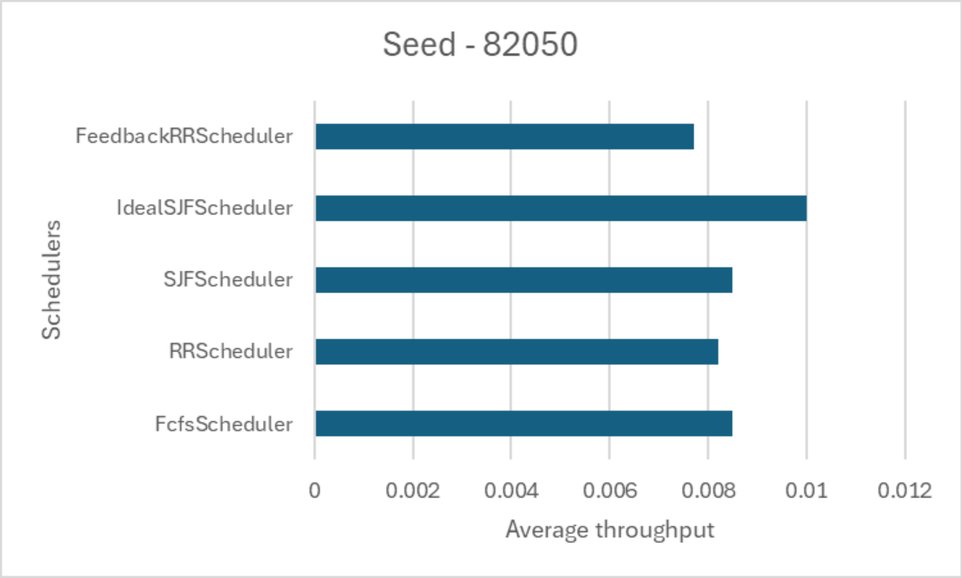

Seed - 5112
Scheduler
FeedbackRRScheduler
IdealSJFScheduler
SJFScheduler
RRScheduler
FcfsScheduler
0
2000
4000
6000
8000
10000
12000
14000
Average waiting time

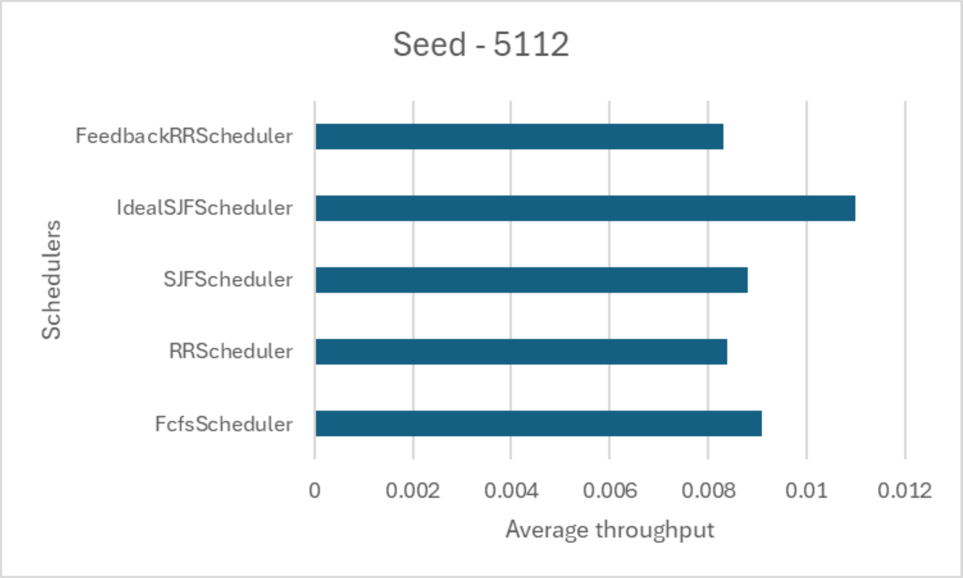

Seed - 97108
Scheduler
FeedbackRRScheduler
IdealSJFScheduler
SJFScheduler
RRScheduler
FcfsScheduler
0
2000
4000
6000
8000
10000
12000
14000
Average waiting time

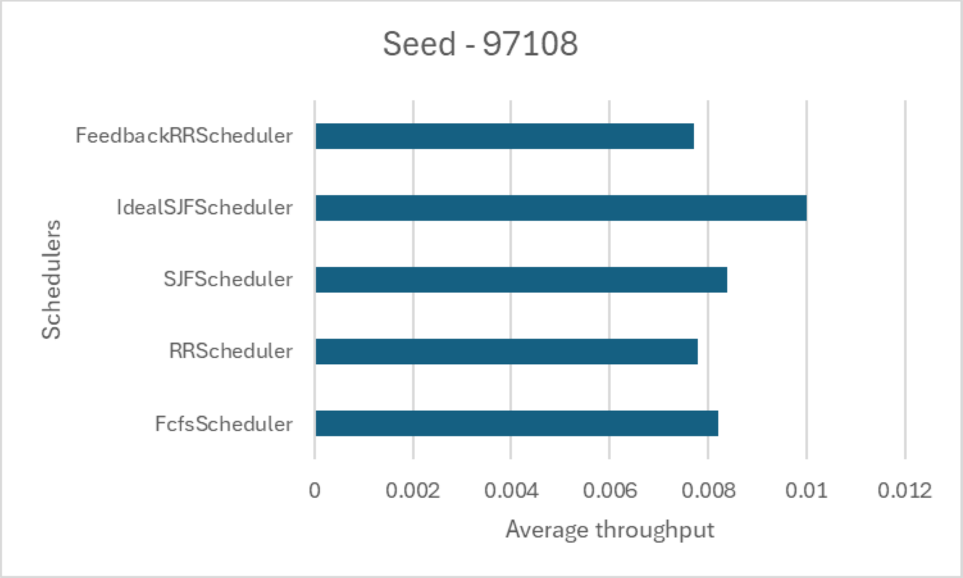
Average throughput from time 0 - 40000



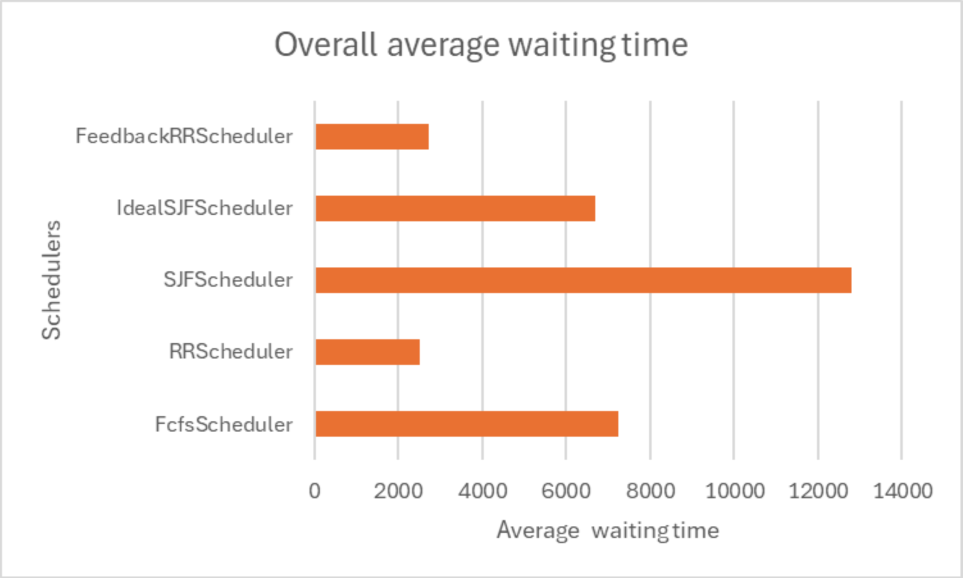


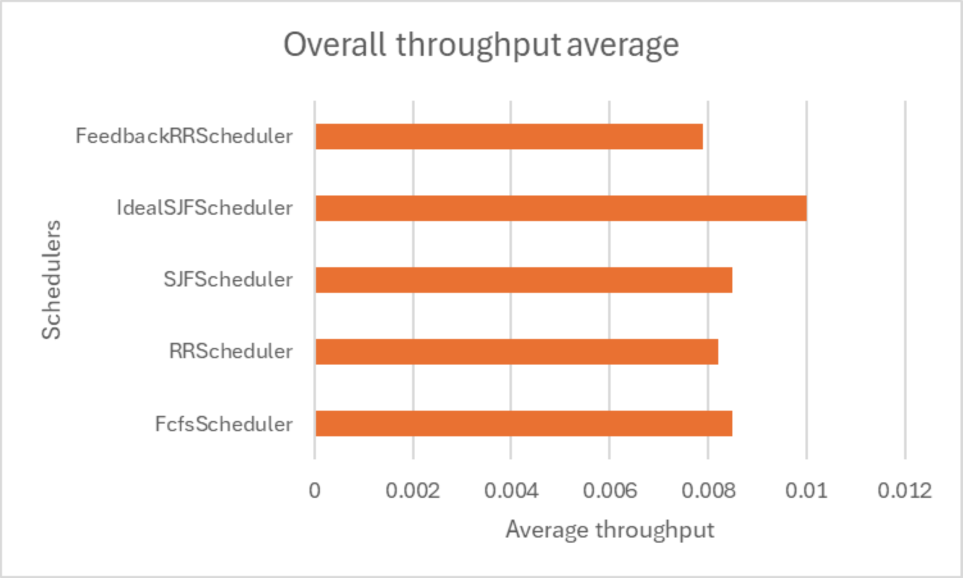



Seed - 5112
Schedulers
FeedbackRRScheduler
IdealSJFScheduler
SJFScheduler
RRScheduler
FcfsScheduler
0
0.002
0.004
0.006
0.008
0.01
0.012
Average throughput


Seed - 97108
Schedulers
FeedbackRRScheduler
IdealSJFScheduler
SJFScheduler
RRScheduler
FcfsScheduler
0
0.002
0.004
0.006
0.008
0.01
0.012
Average throughput

Overall averages





1.4 Discussion

In this experiment I tested two values, and these results are analysed below with a brief description of the possible cause. The throughput is calculated as the time it takes for the experiment, in this case it is 40000, as this is the timeLimit of the experiment and none of the schedulers finished running all processes before reaching the timeLimit.

- **FcfsScheduler**

  Average Waiting Time: Ranges from 6827 to 7679, with an average of 7227.61.

  Average Throughput: Ranges from 0.0077 to 0.0091, with an average of 0.0085

  With FcfsScheduler processing tasks in the order they arrive, this may lead to longer waiting times, especially if longer tasks are scheduled. Throughput may vary depending on the distribution of task lengths and arrival times.

- **RRScheduler**

  Average Waiting Time: Ranges from 2429 to 2575, with an average of 2507.49.

  Average Throughput: Ranges from 0.0078 to 0.0088, with an average of 0.0082

  RRScheduler prevents starvation by allocating a time slice to each process and reducing waiting times for all tasks in the queue. Throughput is relatively good in comparison due to the allocation of time slices.

- **SJFScheduler**

  Average Waiting Time: Ranges from 11996 to 13880, with an average of 12794.59.

  Average Throughput: Ranges from 0.0084 to 0.0092, with an average of 0.0085.

  SJFScheduler prioritizes the execution of shorter tasks first to minimize waiting times. However, if there's a mix of long and short tasks, longer tasks may experience significant waiting times. Throughput is therefore affected by the differences in higher and lower time to completion for different tasks.

- **IdealSJFScheduler**

  Average Waiting Time: Ranges from 6272 to 7007, with an average of 6692.35.

  Average Throughput: Ranges from 0.0102 to 0.0108, with an average of 0.0105

  It has comparatively lower waiting times by accurately predicting task lengths and prioritizing shorter tasks. Higher throughput is achieved due to efficient scheduling of tasks based on the next burst time.

- **FeedbackRRScheduler**

  Average Waiting Time: Ranges from 2409 to 2952, with an average of 2719.91.

  Average Throughput: Ranges from 0.0077 to 0.0079, with an average of 0.0079.

  FeedbackRRScheduler adjusts task priorities based on if they have used their time slice. This adaptability helps reduce waiting times by managing task queues. Throughput varies based on the effectiveness of adjusting task priorities.

Comparing with my hypothesis, the best performing scheduler in terms of average waiting time is the RRScheduler which differs slightly from my previous prediction as the multi-level feedback queue using round robin overall is the second best performing in this metric. Also, in the throughput category I predicted the SJFScheduler, this is not the case as the IdealSJFScheduler has a significantly higher overall result in throughput with 0.0105.

1.5 Threats to Validity

During my experiment there were no threats to the validity of my experiment.

1.6 Conclusion

In this experiment, I aimed to examine the performance of the CPU scheduler under high workload conditions. Through the analysis of the throughput and waiting time, I gained knowledge of how effective these algorithms are when it comes to the scaling up of the schedulers with a high number of processes.

As discussed in the discussion section the best-performing scheduler in terms of waiting time is the RRScheduler. Effectively managing the process queues and preventing starvation by giving each process a time slice.

The scheduler which performed best in having the highest average throughput for the experiment is the IdealSJFScheduler. This can be due to the purely idealized scenario where perfect knowledge of task lengths allows it to be efficient in scheduling the next process.

Overall, these findings show the significance of selecting the correct scheduler depending on what is important in that scenario. If you want a high number of processes completed, then you may select the IdealSJFScheduler but may have a starvation of processes. If you want to prevent starvation then RRScheduler may be a good choice, although this will decrease the number of processes that execute in a certain time.

2. Experiment 2: Investigating the differences in average waiting time for each scheduler.

2.1 Introduction

In this experiment, I investigate how the average waiting time changes for each algorithm and what one performs best when trying to minimise average waiting time. The waiting time is a measure of how long processes stay in the ready queue and for a scheduler to have a small average waiting time they need to have an effective way to switch between different processes to allow processes not to 'idle' in the ready queue for a long amount of time.

The metric can be very important as a high waiting time causes a lot of delay for processes and can drastically decrease the performance of the algorithm. Being able to select the processes with the shortest execute time is effective in this as it means fewer processes must wait, as you get the shorter processes out of the way. If you select a process that has a high execution time it means that all processes in the ready queue must wait for this one to finish. This is usually a lot easier said than done.

The scheduler that I think will be most effective in reducing the average waiting time is the multi-level feedback queue using round robin. As this scheduler is pre-emptive it means that it can be interrupted if a higher priority process enters the queue. In addition, it is round-robin, so it allows for switching out of the processes when it has used the time quantum, this means that a longer process will not stay running for a high amount of time as it will get switched out.

2.2 Methodology

In this experiment, I will use the same input parameters for each output but will be changing the seed each time. In this experiment, it is important to have a lot of processes but with high bursts and I/O times to investigate who can select the correct processes to reduce waiting time. The 'numberOfProcesses' is set to 100, 'meanCpuBurst' of 40, 'meanIoBurst' of 20 and 'meanNumberBursts' of 5. This allows for a good range of processes with different CPU and I/O bursts. All the parameters I have used are below:

numberOfProcesses=100

staticPriority=0

meanInterArrival=12

meanCpuBurst=40

meanIoBurst=20

meanNumberBursts=5

seed=(x)

The parameters for the simulator are below:

scheduler=(y)

timeLimit=10000

interruptTime=5

timeQuantum=15

initialBurstEstimate=40

alphaBurstEstimate=0.5

periodic=false

The names of the input files are the input-seed_x where x is the seed used, in this case, the x follows these numbers (11654, 93333, 24870, 8621, 10291). The output files are named output-seed_x following the same number trend. These output files are under a folder with the name of the scheduler the output is from, the y represents the scheduler. All these files can be found under the folder 2.

I put the initial burst estimate as the same as the CPU mean burst and the time quantum just less than half to allow switching of processes.

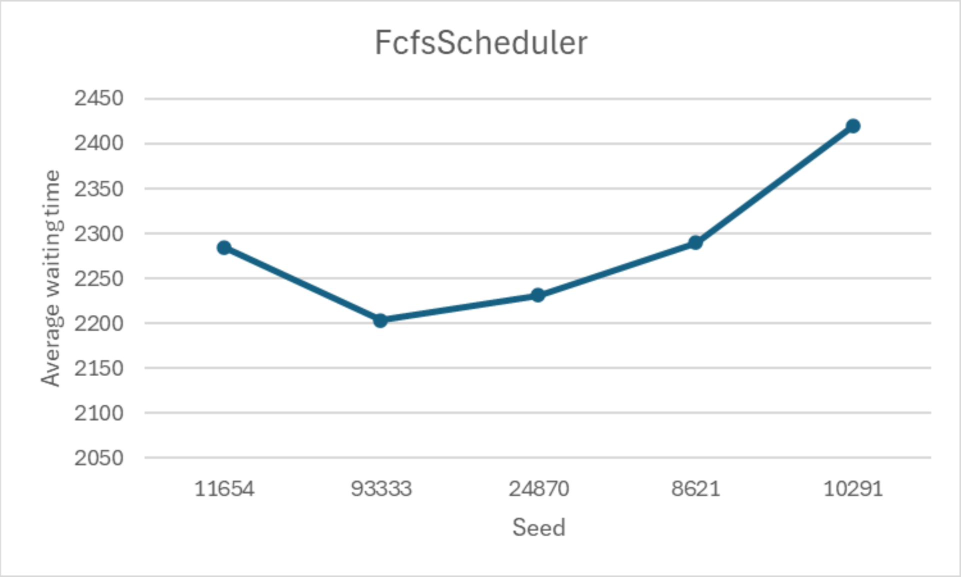
Several processes of 100 allow for a range of processes to be produced using the parameters and to give a good number of results to allow for a fair test.

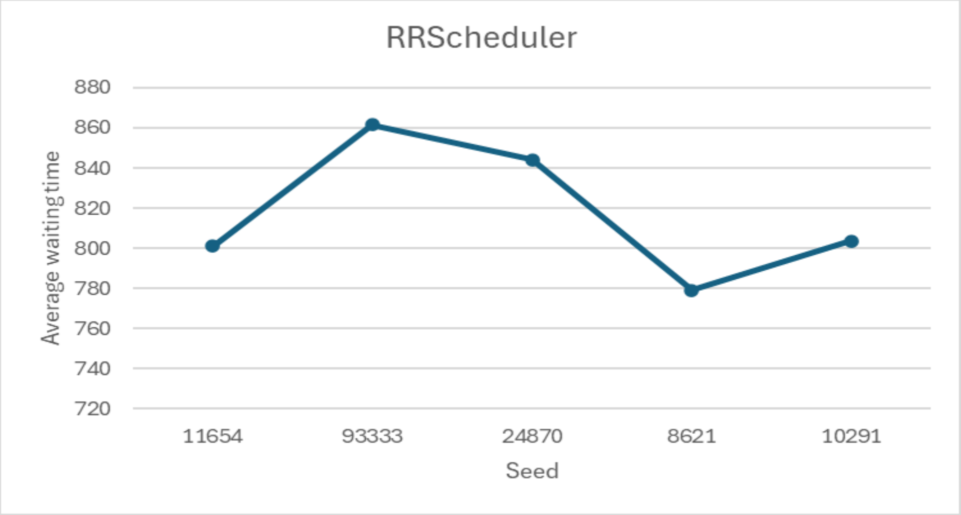
The metric average waiting time is plotted separately with each scheduler with each seed and then the average waiting time is calculated using all these seeds to get a good evaluation of the overall best algorithm when it comes to minimising the average waiting time.

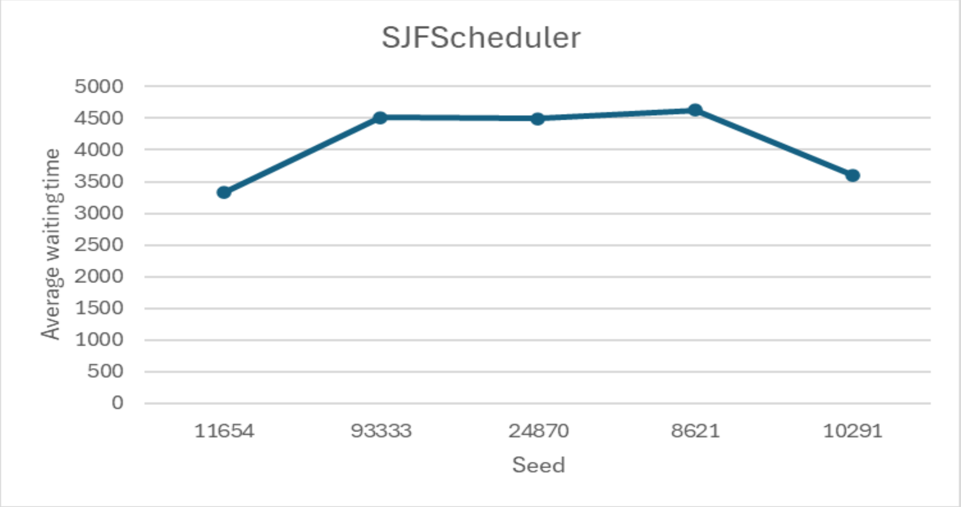
To validate the results, I have used 5 different seeds to compare the average waiting time to ensure a fair test and anomalies are accounted for.

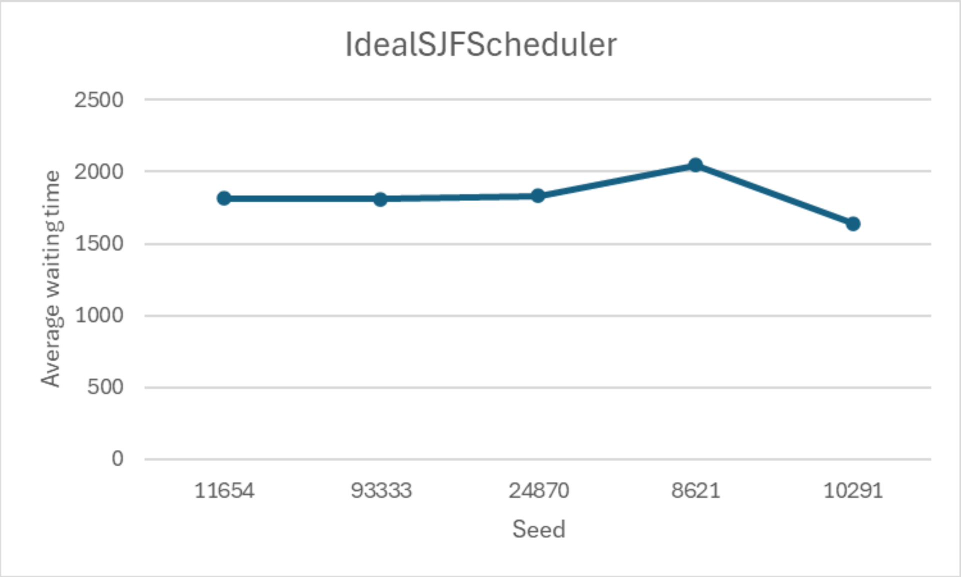
2.3 Results

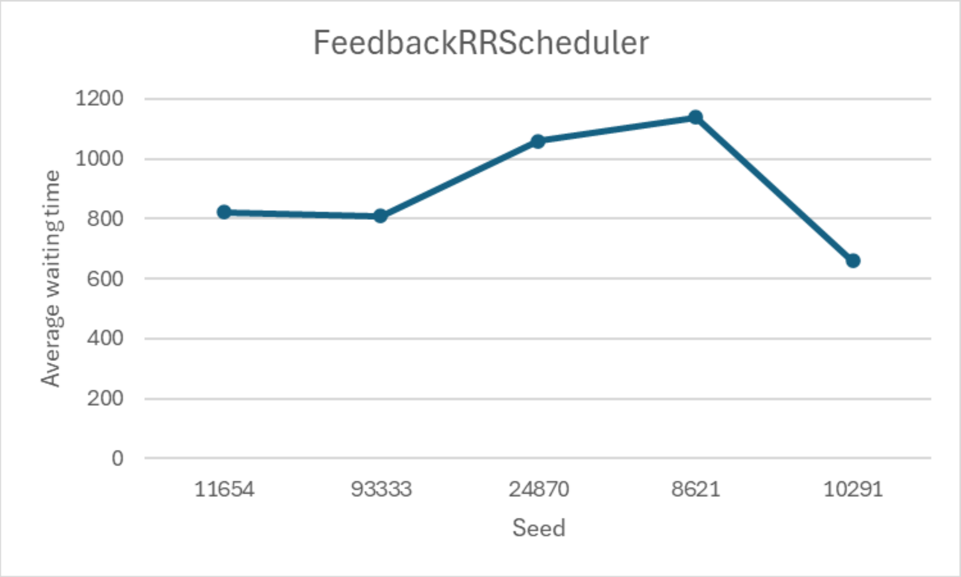
Average waiting time for each seed and scheduler



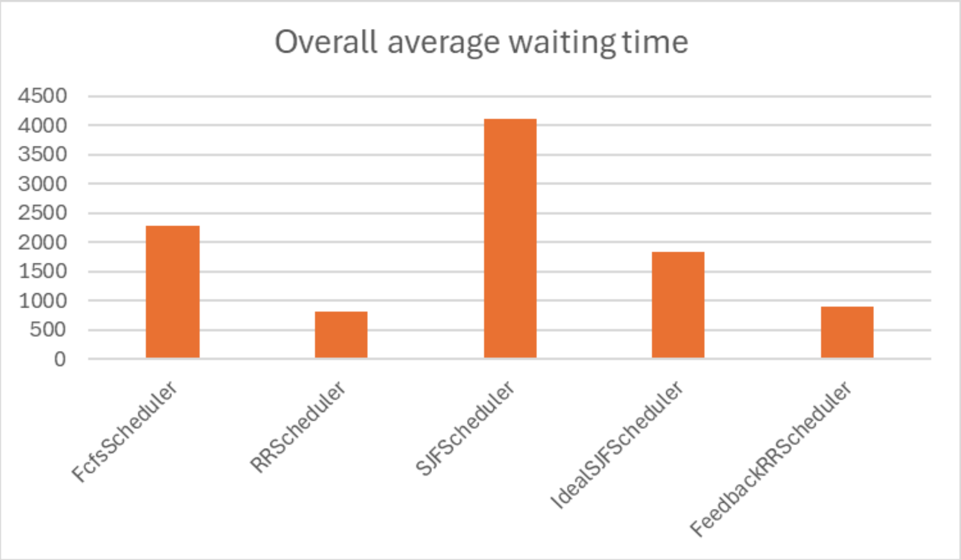








Overall results



2.4 Discussion

Using the results that have been produced from these input and output parameters; let's break down each scheduler's performance based on these results:

- **FcfsScheduler**

  The waiting times across all the seeds have a range of 2203 to 2419 and an overall average of 2285.62.

  The FcfsScheduler has a relatively high waiting time compared to the other schedulers, potentially due to it selecting high burst processes leading to a high average waiting time.

- **RRScheduler**

  The waiting times across all the seeds have a range of 779 to 861 and an overall average of 817.90.

  It has the lowest overall average waiting time but doesn't have the smallest lower bound on the waiting time.

- **SJFScheduler**

  The waiting times across all the seeds have a range of 3329 to 4631 and an overall average of 4113.83.

  These results show a high waiting time in comparison to the other schedulers, which could be due to leaving the higher burst time processes waiting until the end.

- **IdealSJFScheduler**

  The waiting times across all the seeds have a range of 1641 to 2048 and an overall average of 1830.72.

  IdealSJFScheduler represents an idealized version of SJFScheduler, assuming a good prediction of task lengths. It shows better performance compared to SJFScheduler in most cases.

- **FeedbackRRScheduler**

  The waiting times across all the seeds have a range of 659 to 1139 and an overall average of 897.93.

  FeedbackRRScheduler is an enhanced version of RRScheduler, and it demonstrates good performance across different seeds against the RRScheduler.

Overall, SJFScheduler tends to have the highest average waiting times, indicating potential inefficiencies in handling longer tasks; RRScheduler and FeedbackRRScheduler performing relatively better. The FeedbackRRScheduler may not of had the best overall waiting time but had the smallest lower bound on the average waiting time at seed 10291.

2.5 Threats to Validity

A threat to the validity of experiment may be the seed 7885, as this seed gives results that have a high variability compared to the other results. This is especially prominent in the RRScheduler and FeedbackRRScheduler.

2.6 Conclusion

In this experiment, I aimed to investigate the differences in average waiting times between each scheduler and what one performs best when trying to minimize this value and reduce the starvation of the processes in the queue. Starvation is when some processes are not running due to all the other processes that enter the queue having higher priority, this higher priority can be due to the next burst time, predicted burst time or the priority it starts with.

Looking at the results of this experiment I can see that the best-performing scheduler in terms of overall waiting time is the RRScheduler with a value of 817.90, this varies slightly from my hypothesis. There could also be an argument that FeedbackRRScheduler performed better than the RRScheduler due to it having a smaller lower bound on average waiting time and if you removed the seed 7885 (the potential threat to validity) then this scheduler will perform the best.

This experiment helped me get insight into why a smaller waiting time is important in decreasing starvation and promotes fairness in resource allocation to certain processes, this can lead to more efficiency and responsiveness of the system.

3. Experiment 3: Investigating the turnaround time of the schedulers with a high number of bursts.

3.1 Introduction

In this experiment, I will investigate how a high number of bursts of a process can influence which scheduler will be most effective. Having a high number of bursts means having a lot of CPUS and I/O bursts and each scheduler will have a different way of dealing with them.

To measure how well a scheduler can perform with a high number of bursts I will be measuring the average turnaround time as this is a measure of how long it takes from the submission of the process until the time of completion, which includes all of the CPU and I/O bursts.

A scheduler with a lower average turnaround time is more desirable as this shows that the scheduler is highly efficient in dealing with the processes as opposed to something with a higher average turnaround time.

I predict that the scheduler that will perform best with these experiments is the shortest job first (using exponential averaging). This scheduler offers efficient CPU utilization and being able to prioritize the shorter burst times first means that it may lead to lower turnaround times. Also, having a scheduler that can estimate the next burst duration can become crucial and with the exponential averaging this can have a massive help on how well this scheduler can perform.

3.2 Methodology

In this experiment, I will have a high meanNumberBursts, for each input file, of 12 which will help with my investigation. I will be keeping the parameters the same but will be generating different input files using a different seed each time. As discussed, I will be investigating the average turnaround time of each scheduler on a different seed of the input parameters, this will be plotted to make decisions based on the best performing and to compare the results for all the schedulers. The input parameters that I have decided to use are:

numberOfProcesses=50

staticPriority=0

meanInterArrival=12

meanCpuBurst=10

meanIoBurst=10

meanNumberBursts=12

seed=(x)

The parameters for the simulator are below:

scheduler=(y)

timeLimit=10000

interruptTime=1

timeQuantum=5

initialBurstEstimate=10

alphaBurstEstimate=0.5

periodic=false

The names of the input files are the input-seed_x where x is the seed used, in this case, the x follows these numbers (80253, 41649, 72592, 7885, 36334). The output files are named output-seed_x following the same number trend. These output files are under a folder with the name of the scheduler the output is from, the y represents the scheduler. All these files can be found under the folder 3.

I choose the same low number for the CPU and I/O bursts due to the amount of them being 12, I want the main focus of the experiment to be on the number of bursts. Several processes of 50 give a good amount of output data to test.

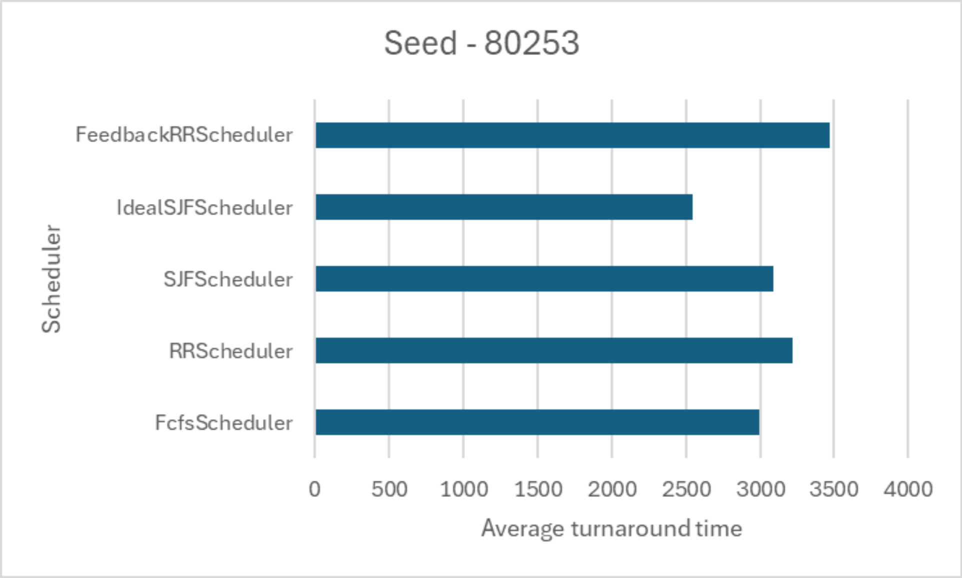
In the scheduler I choose an initial burst estimate the same as the CPU mean burst and a time quantum of half of it so that the schedulers have a chance to swap out processes.

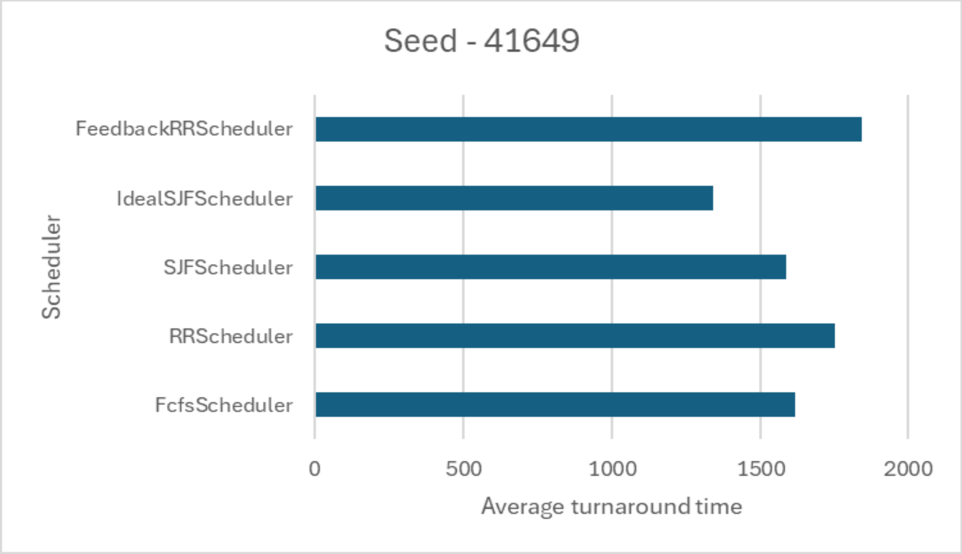
I will run this experiment 5 times with the 5 different input parameters and using this will calculate the average turnaround time for each scheduler on those input files, these will be used to compare the results and have a solid conclusion on the best scheduler. I can also compute an overall average turnaround time using all 5 experiments.

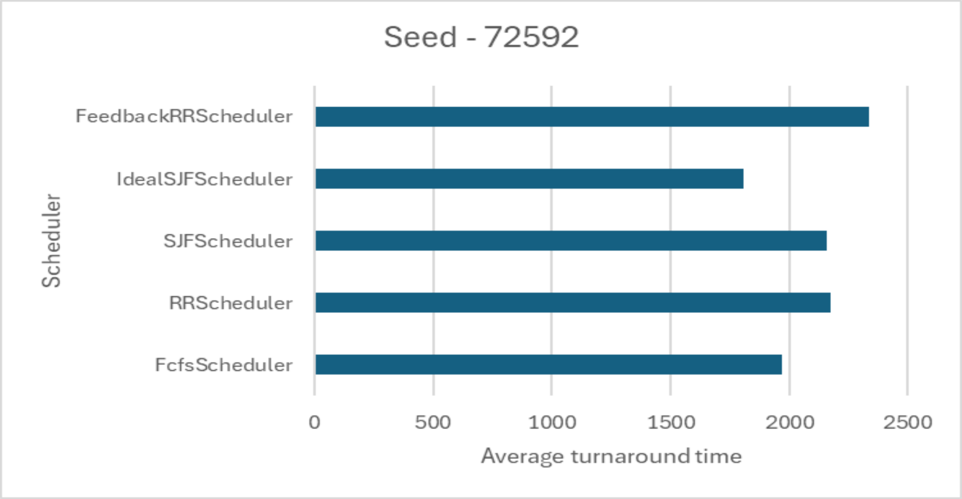
Doing this experiment 5 times allows for any anomalies of the experiment to be hashed out and therefore if one of the input parameters does produce vastly different results than the others I can discard it from the results.

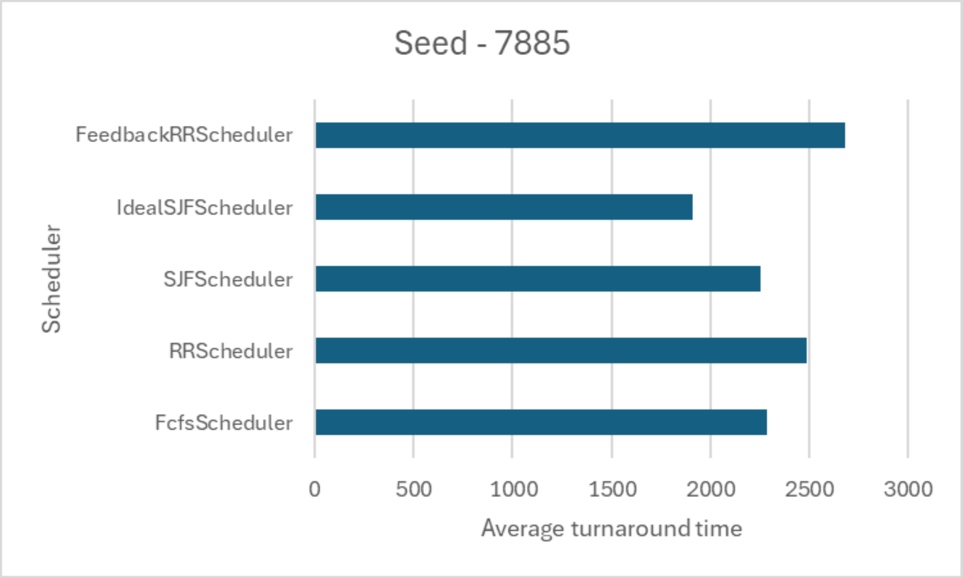
3.3 Results

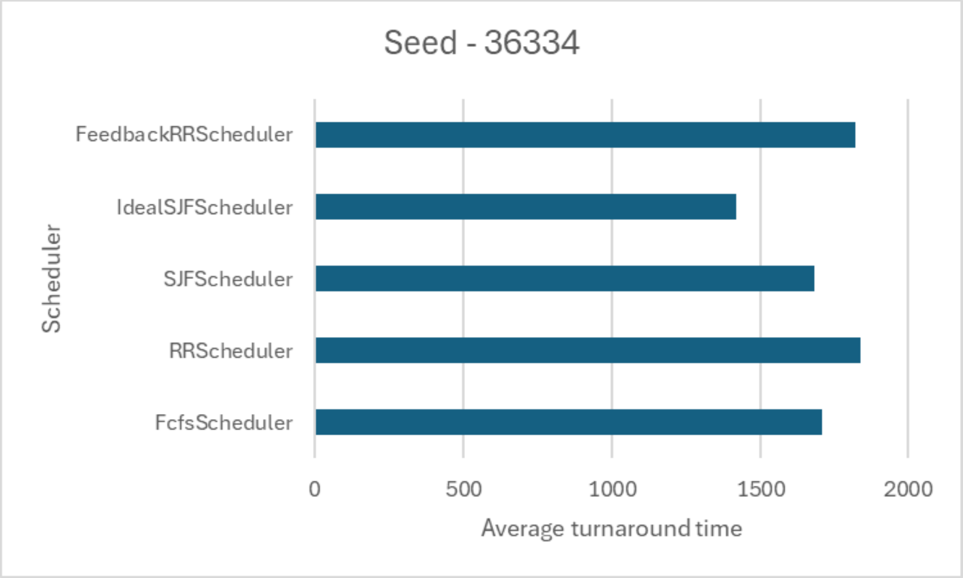
Average turnaround time for each seed



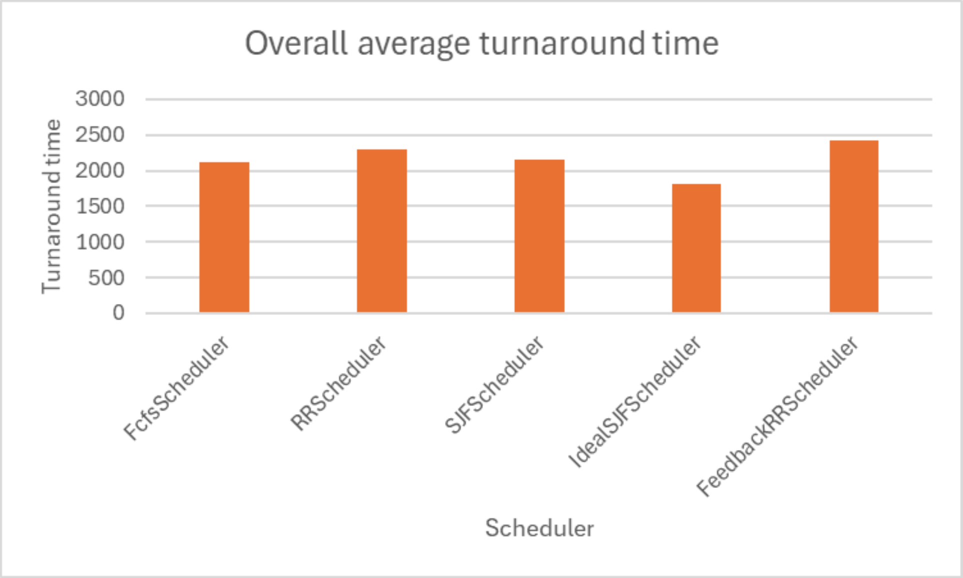








Overall results



3.4 Discussion

Using the average turnaround results that have been produced for this experiment, there is a breakdown of them:

- **FcfsScheduler**

  The turnaround time for all experiments has a range of 1619 to 2991 and an overall average of 2114.75.

  Due to FcfsScheduler processing tasks in the order they arrive, this leads to longer turnaround times, especially if longer tasks are scheduled first.

- **RRScheduler**

  The turnaround time for all experiments has a range of 1753 to 2488 and an overall average of 2295.89.

  RRScheduler provides fairness by allocating equal time slices to each task, but it may not prioritize shorter tasks efficiently, leading to longer turnaround times.

- **SJFScheduler**

  The turnaround time for all experiments has a range of 1588 to 3090 and an overall average of 2155.65.

  This has one of the lowest overall average turnaround results. However, the variability in results suggests that it may not always achieve optimal performance.

- **IdealSJFScheduler**

  The turnaround time for all experiments has a range of 1343 to 2547 and an overall average of 1804.72.

  As an idealized version of SJFScheduler with knowledge of the next burst, IdealSJFScheduler achieves lower turnaround times.

- **FeedbackRRScheduler**

  The turnaround time for all experiments has a range of 1820 to 3466 and an overall average of 2428.53.

  The results suggest that this scheduler does not perform well in terms of turnaround time, with high variability and the highest overall average turnaround time.

Overall, the worst performing scheduler in this experiment is the multi-level feedback queue using round robin. On the other hand, unlike like my hypothesis in the introduction paragraph, IdealSJFScheduler tends to have the lowest average turnaround times, with my prediction of the SJFScheduler having the second-best performance.

3.5 Threats to Validity

During this experiment there were no threats to the validity of the experiment.

3.6 Conclusion

In this experiment, I aimed to explore the impact of a high number of CPU and I/O bursts on the effectiveness of different CPU schedulers. This is measured in the experiment in terms of the average turnaround time.

The results show that the multi-level feedback queue using round robin emerged as the least effective in the average turnaround time, this is due to the swapping out of processes before it has completed, therefore increasing its turnaround time. The most effective in this is the IdealSJFScheduler, which differs from the original hypothesis, with my original prediction being the SJFScheduler. The reason for these being the lowest turnaround time is due to them being able to select the shorter tasks first which gets these processes 'out of the way', but also not neglecting the longer tasks.

From this experiment, I have gained insight into how important turnaround is in ensuring the timely completion of tasks and enhancing system responsiveness. Also, when implemented in a real-life scenario and increases user satisfaction as it leads to faster response time.