Shashank P
200010048

CS314 Operating Systems Lab
Lab 4

1 Scheduling Schemes

To implement the scheduling scheme, I have assumed that after every CPU burst, the process goes into a blocked state. This will allow another CPU-bound process to execute while the I/O device runs in parallel. To simulate this, I have used the following:

1. The environment is updated every **1** unit
2. Maintained **2 priority queues** one for CPU and another for I/O device
3. I/O device never preempts, while CPU processes can preempt for SRTF

Metrics such as turnaround time, waiting time, response time, system penalty ratio and system throughput were used. To execute the program:

- Run **make** command in root directory.
- A **bin** folder will be created with **SJF** and **SRTF** executable files.
- Run any algorithm with the path to the test file as a command line argument.
- The output will be displayed in the terminal also a CSV file of individual metrics will be stored in the test file directory.

1.1 Shortest Job First (SJF)

In this algorithm, the job with the shortest burst time is prioritised and executed first. I have implemented **Non-Preemptive** Shortest Job First algorithm. For this, whenever a new process is incoming, it is first added to the CPU priority queue, which keeps the lowest burst time at the top. The following steps are performed,

1. At every time step **(1 unit)** check all incoming processes and add them to the CPU-ready queue.
2. If the CPU is empty, a process is added to the CPU from the CPU-ready queue.
3. If the IO device is empty, a process is added from the IO queue.
4. If CPU is not empty, run it for **1** time-step. Similarly, for the IO device.
5. If a part of the process is completed in CPU, move the process to the IO queue, and vice-versa.
6. If the burst time is -1, remove the process entirely. Do this until all processes are completed.

1.2 Shortest Remaining Time First (SRTF)

In this algorithm, the job with the shortest remaining time is prioritised and executed first. This is a **preemptive** scheduling algorithm. Since it is mentioned that the I/O device fulfils the request sequentially, the SRTF scheduling was only applied to CPU jobs, while I/O jobs still follow SJF scheduling.

1. At every time step **(1 unit)** check all incoming processes. Before adding them to the CPU-ready queue, check whether it's remaining time is less than the current CPU's process. If yes, preempt and store the old process in the queue instead.
2. If the CPU is empty, a process is added to the CPU from the CPU-ready queue.
3. If the IO device is empty, a process is added from the IO queue.
4. If CPU is not empty, run it for **1** time-step. Similarly, for the IO device.
5. If a part of the process is completed in the CPU, move the process to the IO queue. For the other case, perform the same preemption test mentioned in point 1.
6. If the burst time is -1, remove the process entirely. Do this until all processes are completed.

2 Characteristics

Here are a few distinct differences between both the scheduling algorithms.

Shortest Job First	**Shortest Remaining Time First**
Non-Preemptive	Preemptive
Involves less number of context switches	More number of Context switches
Response time is high for short processes	Response time is 0 for consecutive short processes
Short processes are executed first	Short processes execute fast, while long processes are stalled

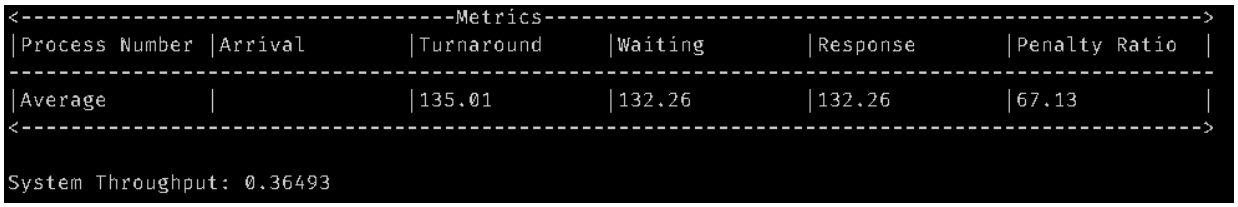
Some disadvantages of both include

1. Both are infeasible as burst time is not known upfront.
2. Both lead to starvation of long-running jobs.
3. Both algorithms are the same once all processes are added to the CPU-ready queue.

3 Case specific analysis

I generated a test case where the first process arrives at 0 with a burst of 50, followed by around 70 processes of burst 2 arriving at 1. Part of the file is shown below.

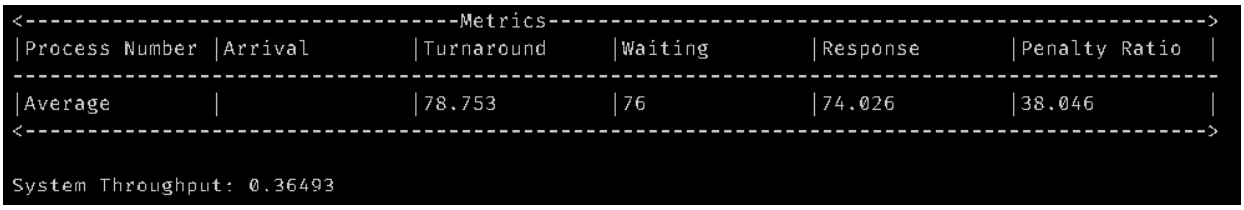
```
0 50 -1
1 2 -1
1 2 -1

 Repeat...70

1 2 -1
1 2 -1
```



Process Number	Arrival	Turnaround	Waiting	Response	Penalty Ratio
Average		135.01	132.26	132.26	67.13

System Throughput: 0.36493

Figure 1: SJF Algorithm



Process Number	Arrival	Turnaround	Waiting	Response	Penalty Ratio
Average		78.753	76	74.026	38.046

System Throughput: 0.36493

Figure 2: SRTF Algorithm

We obtain the shown metrics when running both algorithms on the above process list.

We see that in the case of **SJF**, all the average metrics values are very high as it is non-preemptive. It runs only jobs with a very high burst time before moving on to other small burst time jobs. Therefore, in this case, the preemptive **SRTF** algorithm works very well, as shown in the metrics.

4 Disadvantage of SRTF

One of the disadvantages of **SRTF** is that it can lead to starvation of higher CPU burst jobs. This can increase the waiting and turnaround time for high burst time jobs compared to the lower ones. Ex. In the above case, the job with a burst time of 50 is starved while all other smaller jobs are run, even though it came first.

5 Plots

I obtained the following plots by running the algorithms on the given test cases. From the plots, we can see that the given test cases **SRTF** outperform SJF. This is because **SRTF** is more powerful than **SJF** as it can preempt whenever a lower burst process arrives. System throughputs were the same for both **SRTF** and **SJF**.

Test Case No.	System throughput
Test 1	0.004902
Test 2	0.018614
Test 3	0.0057416

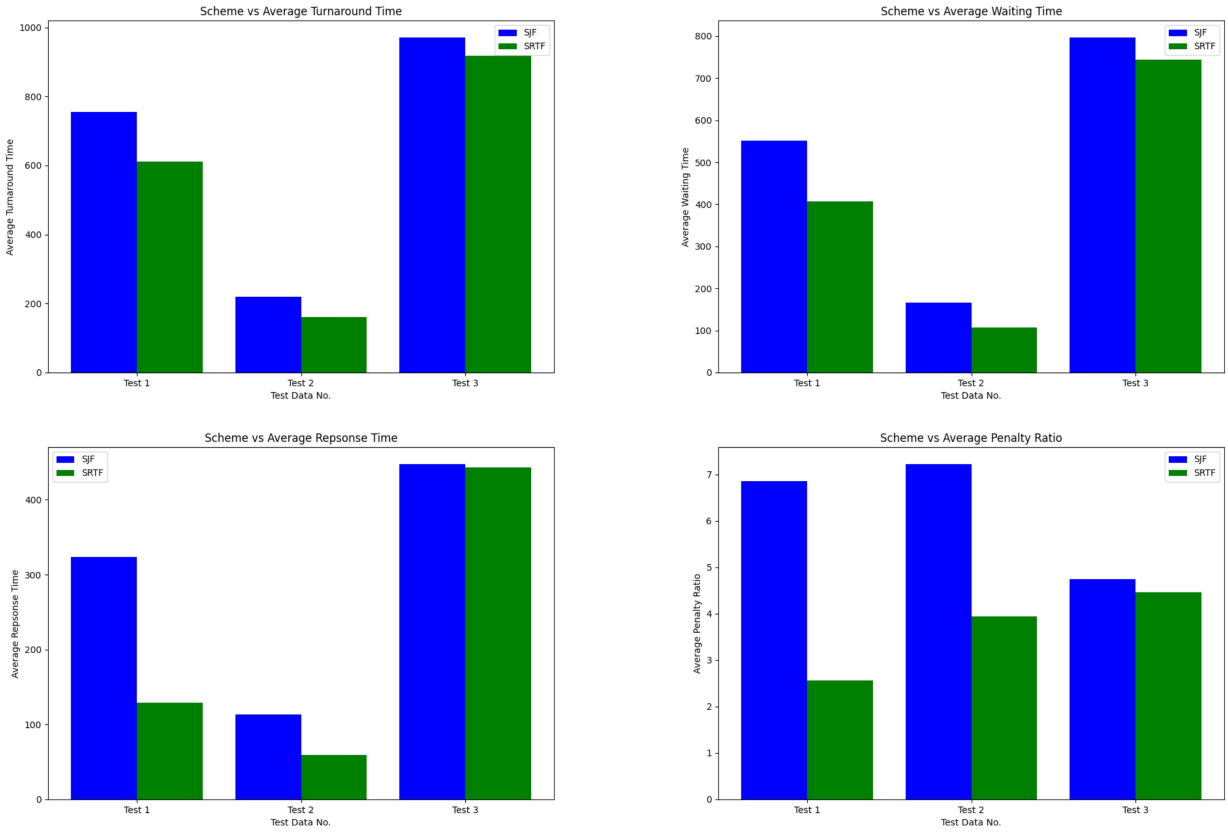


Figure 3: Metrics