Harrithha
200010018

CS314 Operating Systems Lab
Lab 4

---

1 Scheduling Schemes

For implementation of the algorithms, it is assumed that after every CPU burst, the process goes into a blocked state while the I/O operation is supposed to be performed. This will allow another CPU-bound process to execute while the I/O device runs in parallel. It is simulated as follows:

- Two queues are maintained - one for **CPU bound** processes and another for **IO bound** processes.
- IO never preempts where as the CPU queue can preempt processes depending on the algorithm.
- A variable **currentTime** is used to simulate the time, which is updated every one unit.
- The required metrics: **turnaround time, waiting time, response time, system penalty ratio and system throughput** were calculated to measure performance.

Code setup is as follows:

- On running **make** in the directory, 2 .out files namely, **./sjf.out** and **./srtf.out** will be created.
- Usage is as follows: `./sjf <path_to_data_file>` or `./srtf <path_to_data_file>`.

1.1 Shortest Job First (SJF)

The shortest job first algorithm (**non-preemptive**) initially sorts processes which have arrived (arrivalTime $\leq$ currentTime) in the CPU queue according to the burst time. Then:

- Once the CPU burst of a particular process is completed, it will be removed from the CPU Queue and the IO bound process following it will be added to the IO Queue.
- Once the IO bound process is completed, it will be removed from the IO Queue and the next CPU bound process will be added to the CPU Queue.
- If an entire process is completed, that is we reach **-1** as given in the input file, the last CPU bound process that just completed will be removed from the CPU Queue and the CPU Queue will be sorted to run the process which has least burst time among the arrived processes.
- **currentTime** is incremented by **1**.

1.2 Shortest Remaining Time First (SRTF)

The shortest remaining time first algorithm (**preemptive**) initially sorts processes which have arrived (arrivalTime $\leq$ currentTime) in the CPU queue according to the burst time. Then:

- While a CPU bound process is being executed, at each second (while updating value of **currentTime**), we check whether a new CPU bound process has arrived. If yes, we preempt the current process if the remaining burst time of the current process is greater than the burst time of the new incoming process.
- Once the CPU burst of a particular process is completed, it will be removed from the CPU Queue and the IO bound process following it will be added to the IO Queue.
- Once the IO bound process is completed, it will be removed from the IO Queue and the next CPU bound process will be added to the CPU Queue and the cycle follows from Step 1.
- If an entire process is completed, that is we reach **-1** as given in the input file, the last CPU bound process that just completed will be removed from the CPU Queue and the CPU Queue will be sorted to run the process which has least burst time among the arrived processes.
- **currentTime** is incremented by **1**.

2 Expected Job Characteristics

2.1 SJF

- Greedy, Non-preemptive.
- Involves less number of context switches.
- Response time tends to be high.

2.2 SRTF

- Greedy, Preemptive.
- More number of Context switches.
- Response time is 0 if consecutive processes have decreasing burst times.
- Shorter processes are executed fast.

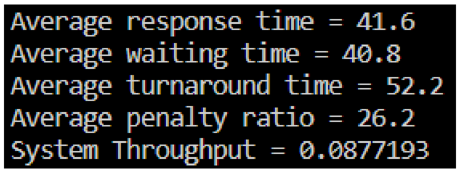
Both are **suitable** in the following cases:

- Short execution times, else may cause starvation.
- High predictability of execution times.
- Low variability in execution times.
- Fewer processes.
- Once all the processes are in the CPU Queue, both algorithms are the same.

3 Algorithm Comparison

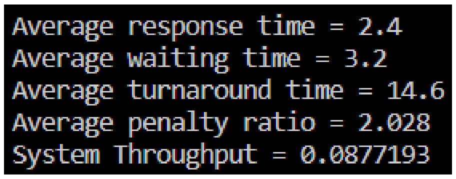
After trying out various test cases, I observed that **SRTF** always performed **better** than SJF as shown for the below test case:

```
0 30 -1
1 2 -1
1 2 -1
1 2 -1
2 1 -1
```



```
Average response time = 41.6
Average waiting time = 40.8
Average turnaround time = 52.2
Average penalty ratio = 26.2
System Throughput = 0.0877193
```

Figure 1: SJF Results



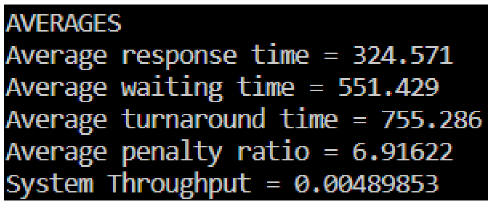
```
Average response time = 2.4
Average waiting time = 3.2
Average turnaround time = 14.6
Average penalty ratio = 2.028
System Throughput = 0.0877193
```

Figure 2: SRTF Results

This is because SJF is nonpreemptive. Thus, it runs a long job which arrived first, to completion even though shorter jobs arrive later and can be completed fast.
SRTF Disadvantage: Even though a long job came first, it will be preempted if shorter jobs keep coming, which leads to a high value of completion and turnaround time.

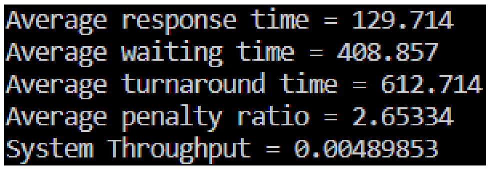
4 Performance Analysis

After running the test data, I observed the following for **process1.dat**:



```
AVERAGES
Average response time = 324.571
Average waiting time = 551.429
Average turnaround time = 755.286
Average penalty ratio = 6.91622
System Throughput = 0.00489853
```

Figure 3: SJF Results



```
Average response time = 129.714
Average waiting time = 408.857
Average turnaround time = 612.714
Average penalty ratio = 2.65334
System Throughput = 0.00489853
```

Figure 4: SRTF Results

Similarly in the **process2.dat** and **process3.dat**, I observed high values for all metrics in case of **SJF** as compared to SRTF except **system throughput**, which is approximately the same for SJF and SRTF as shown below.

Process	**System Throughput**
1	0.00489
2	0.0186
3	0.0057

5 Plots

The above analysis can be easily verified by observing the following plots:

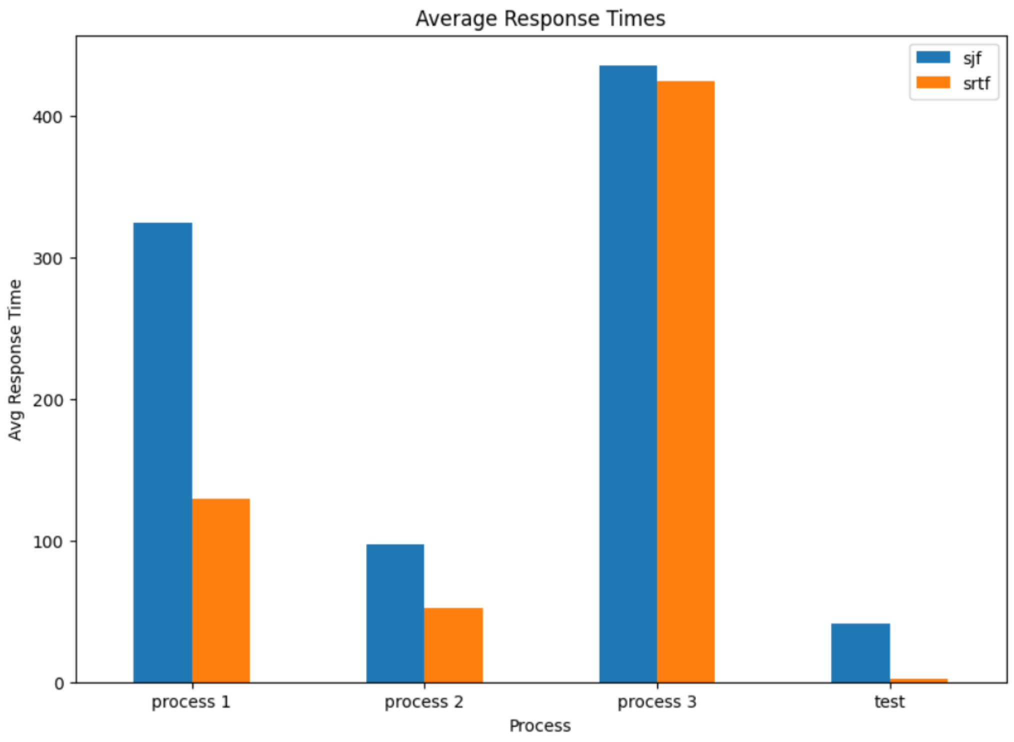


Figure 5: Response Time Comparison

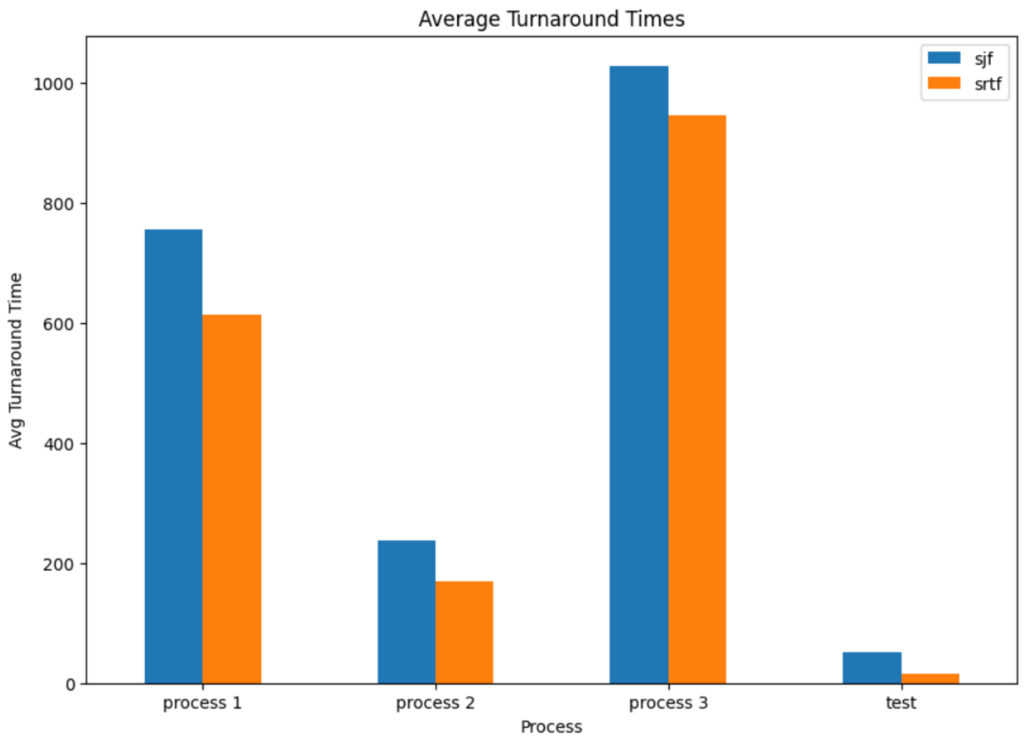


Figure 6: Turnaround Time Comparison

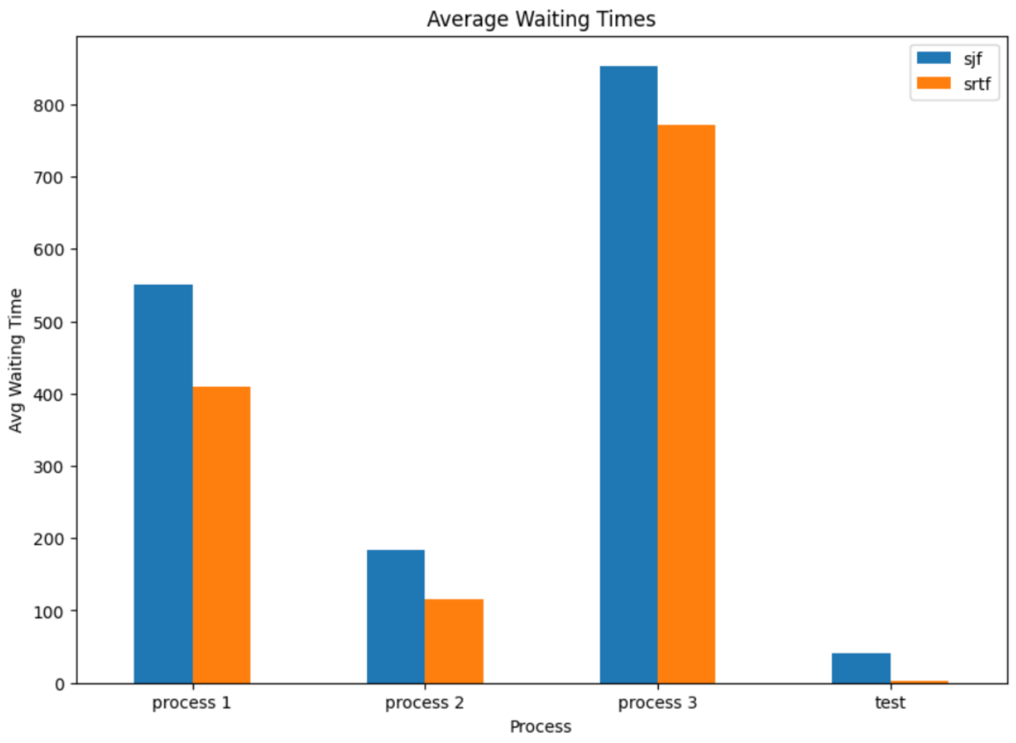


Figure 7: Waiting Time Comparison

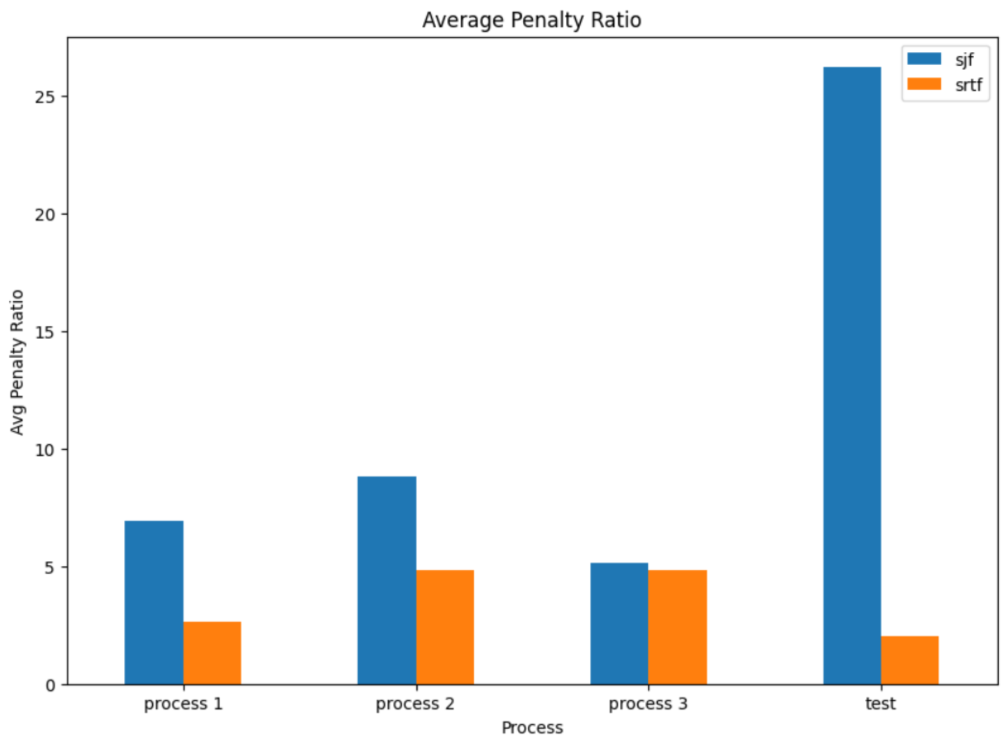


Figure 8: Penalty Ratio Comparison

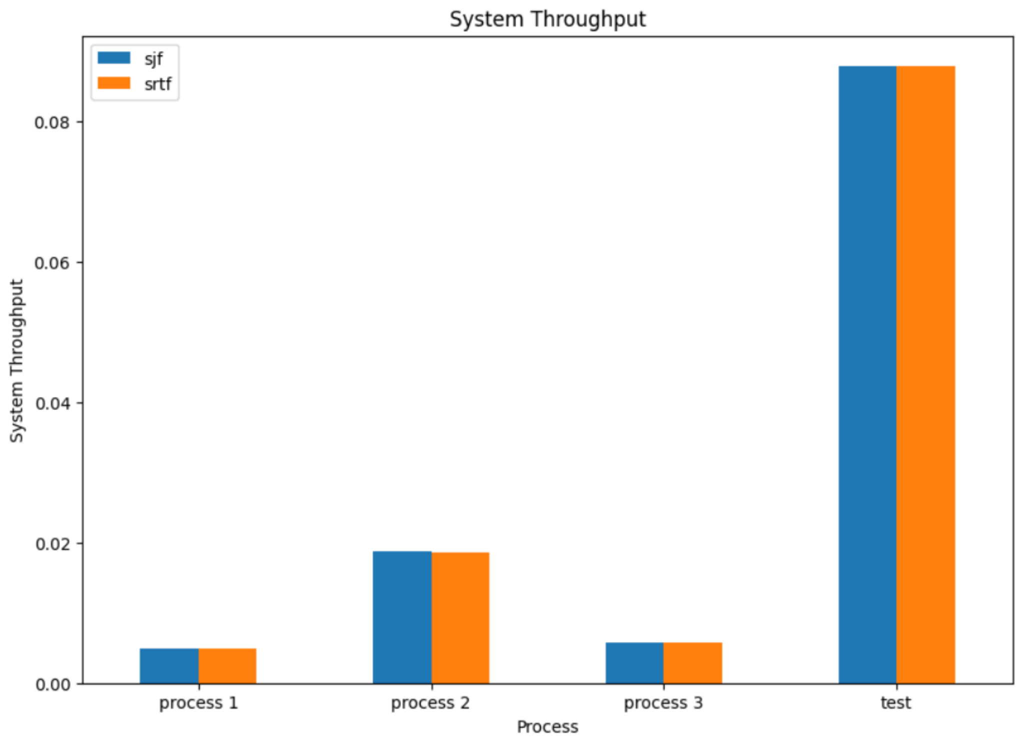


Figure 9: System Throughput Comparison