CS314: Lab Report
Assignment 4 - SJF and SRTF Schedulers

Devdatt N
200010012@iitdh.ac.in

31 January 2023

1 Part I - Shortest Job First Scheduler

1.1 Brief Overview

The shortest-job-first scheduler has been done in C++ and employs a class-driven approach where each process is a process object. The SJF algorithm uses a variety of data structures from process vectors to priority queues for CPU bursts. A system clock variable is incremented in parallel to keep track of time in the simulation.

In the SJF scheduler, when the CPU is idle, the job in the ready queue with the lowest CPU burst time is chosen for execution. We have to also note that the SJF scheduler is not a pre-emptive scheduler, which means that a job which is running on the CPU has to complete before another can come in.

1.2 Expected job characteristics

These are the expected job characteristics of the SJF scheduler:

- Priority-based: SJF uses burst time as the priority for selecting the process to be executed, with the shortest burst time having the highest priority.
- Non-preemptive: A non-preemptive SJF scheduling algorithm does not allow for interruptions, and a process will continue to execute until it finishes or is blocked.
- Optimizes for throughput: SJF is optimized for maximizing throughput, as the shortest processes are executed first, leading to a reduced overall waiting time for all processes.
- Not suitable for real-time systems: SJF is not suitable for real-time systems, as it does not provide guaranteed completion time for processes.

1.3 Testing for various inputs

Since the program is written in C++, we use the g++ compiler to build the executable. Specifically, we use the command `g++ SJF.cpp -o SJF` .

We need only to pass the input file to the executable. To test, we run `.SJF.exe .input.dat > output.txt` where `input.dat` is our input file and `output.txt` is where we want our stats to be saved to. The `>` operator redirects the standard output to the file stream.

`output.txt` will now contain the details for every process, including turnaround time, waiting time, and penalty ratio. However, we only need the overall summary calculated at the end and included at the end of the file.

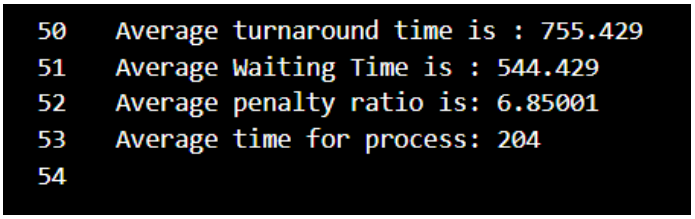

Figure 1: Results for first input

We can see that the average turnaround time here is `221.611` , the average waiting time is `163.611` , the average penalty ratio is `7.38` and the average time taken for completion of a process is `53` . Similarly, we obtain these results for the second and third inputs respectively:

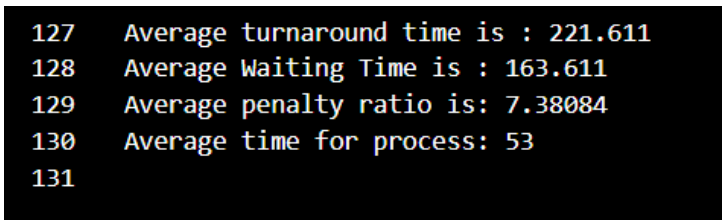

Figure 2: Results for second input

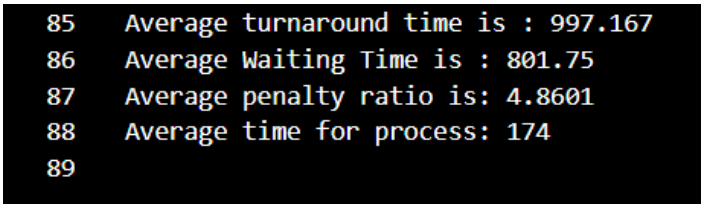

Figure 3: Results for third input

Here, it is important to note that the penalty ratio was obtained as a whole over multiple bursts of the same process for simplicity.

Moreover, from the process-specific outputs which we have not included in this report because of the length, which however are included in the zip file, we can understand that processes with shorter bursts have a lower turnaround time and are successful in gaming the SJF scheduler.
At the same time, we can also observe that processes with longer burst values are starved by the SJF scheduler.

1.4 Analyzing the SJF Scheduler

As we know, the SJF scheduler works by choosing the shortest job which can be taken first. In our simulation, we do this by having a priority queue for our cpu burst queue, where processes with lower burst times have a higher priority.

This being said, we can predict with confidence that SJF works best when processes arrive at the same time. This is because if a process with a pretty high burst time arrives at an earlier time, it might hog the cpu and all the other processes will be starved. Therefore, a dataset where SJF works well would look something like this:

Coming to the output for this, we can see that the penalty ratio is quite low at 2.32 : Of course, we need to compare this with a case where SJF may not be so suitable. As we discussed before, such a case can occur when a long job arrives at an earlier time and hogs the cpu. This basically means that a dataset with different arrival times is bad for SJF. Such a dataset would look something like

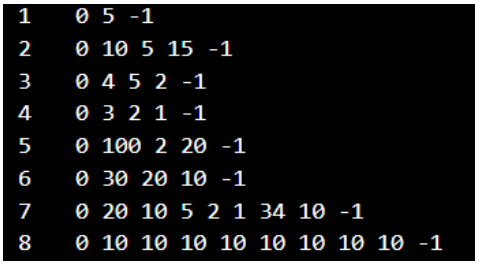

Figure 4: A dataset well-suited for SJF

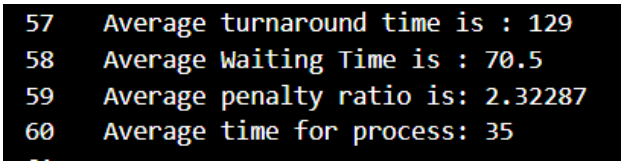

Figure 5: Results

this: Let us look at the results for this dataset: As we can see, the results are horrible- the penalty

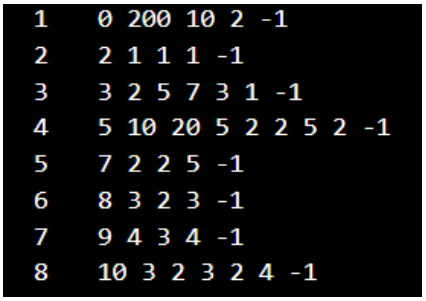

Figure 6: A dataset which is not suited for SJF

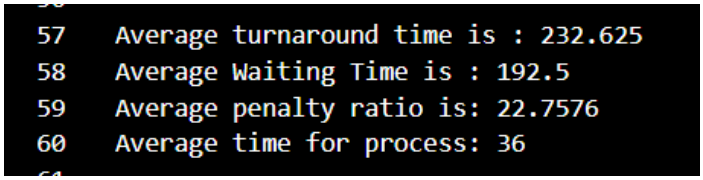

Figure 7: Results

ratio is at 22.75, which means SJF is not at all optimized for this scenario. In this case, SJF lets the first process take up all the CPU time and does not let the shorter processes which come after get any CPU time.

Now, it is necessary that we compare the SJF scheduler to the SRTF scheduler to actually know which fares better and by how much.

2 Part II- Shortest Remaining Time First Scheduler

2.1 Overview and Implementation

The shortest-remaining-time-first scheduler has been implemented in a very similar fashion to the SJF scheduler, except for some key details. In the SRTF scheduler, for pre-emption, we have to simulate a sort of time slice. Since we do not know what time slice would be optimal, we leave it to the user to decide the same. The input file and time slice as well are taken as arguments.

2.2 Expected job characteristics

The following are the expected job characteristics for the SRTF scheduler:

- Preemptive: SRTF is a preemptive scheduling algorithm, which means that a process with a shorter remaining time can interrupt a process that is currently executing.
- Dynamic: SRTF is a dynamic scheduling algorithm as it updates the remaining time for each process based on the amount of time it has executed and the amount of time remaining.
- Priority-based: SRTF uses remaining time as the priority for selecting the process to be executed, with the shortest remaining time having the highest priority.
- Optimizes for turnaround time: SRTF is optimized for minimizing turnaround time, as the process with the shortest remaining time is executed first, leading to a reduced overall waiting time for all processes.
- Suitable for real-time systems: SRTF can be used in real-time systems, as it provides a guaranteed minimum completion time for processes. However, it may lead to process starvation if long processes are constantly interrupted by shorter processes.

2.3 Testing for various inputs

We run `g++ SRTF.cpp -o SRTF` followed by `SRTF.exe input.dat 5 > output.txt` . Here, 5 signifies that we are setting the time slice to 5 units. Of course, we will compare this later on to other time-slice settings to see how everything goes. Comparing this with the results of the first input

```
50   Average turnaround time is : 628
51   Average Waiting Time is : 417
52   Average penalty ratio is: 2.64911
53   Average time for process: 204
54
```

Figure 8: Results for first input

from SJF, we can see that turnaround time is much lesser than SJF, where SRTF gives an average turnaround time of 628 while SJF gives around 755.
Similarly, average waiting time is better as well, where SRTF gives 417 and SJF gives around 544. Furthermore, the penalty ratio for SRTF is much lesser, where SRTF has only 2.64 compared to 6.85 that SJF has. This means that for this particular dataset, SRTF was much better optimized.

Now, we can compare with other inputs as well:

```
127   Average turnaround time is : 158.722
128   Average Waiting Time is : 100.722
129   Average penalty ratio is: 3.74716
130   Average time for process: 54
131
```

Figure 9: Results for second input

For the second input, we can see similar results as well. SRTF fares considerably better. For the third input, however, we can see that SRTF and SJF have almost the same outcomes, even though SRTF has a slight edge. This might be because of the type of inputs in the third input file combined with the time slices we entered. We can verify this by entering a time slice of 20 instead: We can see that the turnaround time, waiting time and penalty ratio has lessened, even though not much.

```
85   Average turnaround time is : 955.083
86   Average Waiting Time is : 759.667
87   Average penalty ratio is: 4.71331
88   Average time for process: 174
89
```

Figure 10: Results for third input

```
85   Average turnaround time is : 940.333
86   Average Waiting Time is : 744.917
87   Average penalty ratio is: 4.65275
88   Average time for process: 174
89
```

Figure 11: Time slice changed for third input

The difference between SJF and SRTF schedulers is not much in this case, probably because of the uniformity of remaining time while processes are running. It is also important to understand that average time for process for SJF and SRTF schedulers are same because system throughput remains the same.

2.4 Further analyzing the SRTF scheduler

It is necessary that we understand where our SRTF scheduler performs the best and where it does not do so well. In order to do so, we must think of the reason why SRTF was introduced- it was so as to fix the issues SJF had with processes arriving at different times. Therefore, we can predict that processes which arrive at different times will work great with the SRTF scheduler, where the time slice for the pre-emptive check must be the average time between which two processes arrive.
Quite naturally, such a dataset can look something like this: This is the exact dataset which we used

```
1   0 200 10 2 -1
2   2 1 1 1 -1
3   3 2 5 7 3 1 -1
4   5 10 20 5 2 2 5 2 -1
5   7 2 2 5 -1
6   8 3 2 3 -1
7   9 4 3 4 -1
8   10 3 2 3 2 4 -1
```

Figure 12: A dataset which is suited for SRTF

for proving that SJF does not work so well with different arrival times. Let us try running SRTF on this with a timeslice of 1: As we can see, the penalty ratio is incredibly good, at 1.91, as opposed to

```
57   Average turnaround time is : 61.875
58   Average Waiting Time is : 21.75
59   Average penalty ratio is: 1.91113
60   Average time for process: 34
```

Figure 13: Results

the 22.75 which we got while running SJF on this dataset. This proves that SRTF works quite well with processes with different arrival times.
However, SRTF might not run as good as before if the processes all arrive at the same time. In this

case, SRTF might keep on switching between processes with the same arrival time, which might end up in the processes having a bad turnaround time. This can be demonstrated by running the following dataset:

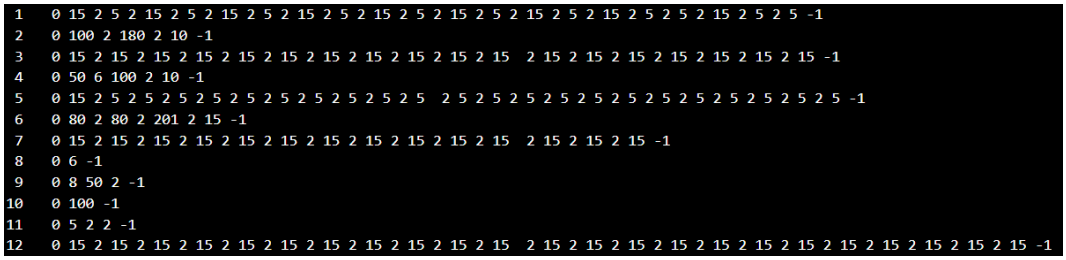

Figure 14: This dataset might not be suitable for SRTF

Running SRTF for this dataset, we obtain this result: Here, we have used the time slice 1. We are not

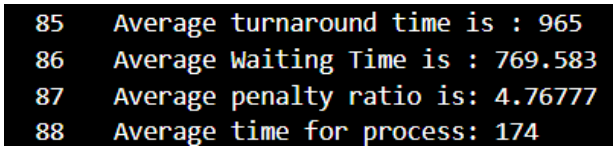

Figure 15: Result 1

worrying about process switch costs here. SRTF returns a slightly worse penalty ratio than usual, 4.76. The fault of SRTF lies in the fact that we cannot find an appropriate time slice for pre-emption and this might mean many processes will have very high turnaround times. Digging deeper:

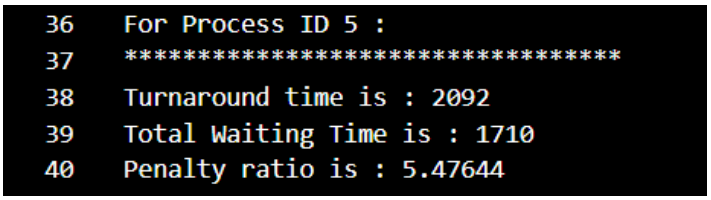

Figure 16: Result 2

We can find that this process has a turnaround time of more than 2000, which is more than twice the average turnaround time. Moreover, it has spent most of its time in waiting while other processes had to wait less. Another example would be:

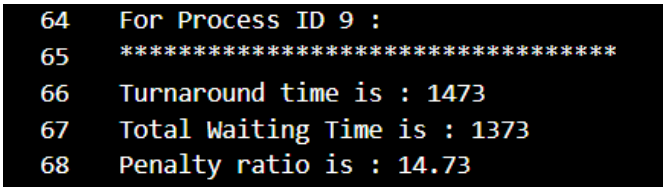

Figure 17: Result 3

Here, we can see the penalty ratio has gone up to 14.71. Therefore, we can safely assume that even with SRTF, when many processes come at the same time, it is not only difficult to determine the time slice since process bursts might be variable, but also that SRTF does not solve the problem of many processes being starved.

3 Graphical Analysis

Now, let us compare between SJF and SRTF schedulers using the average turnaround time, the average waiting time and penalty ratio respectively. Here, the time slice used for pre-emption in the SRTF scheduler is 5. Furthermore, Suit stands for a dataset for which a particular scheduler performs well and Shot stands for a dataset where a particular scheduler has shortcomings. Let us look at the plotted graphs now.

3.1 Turnaround Time

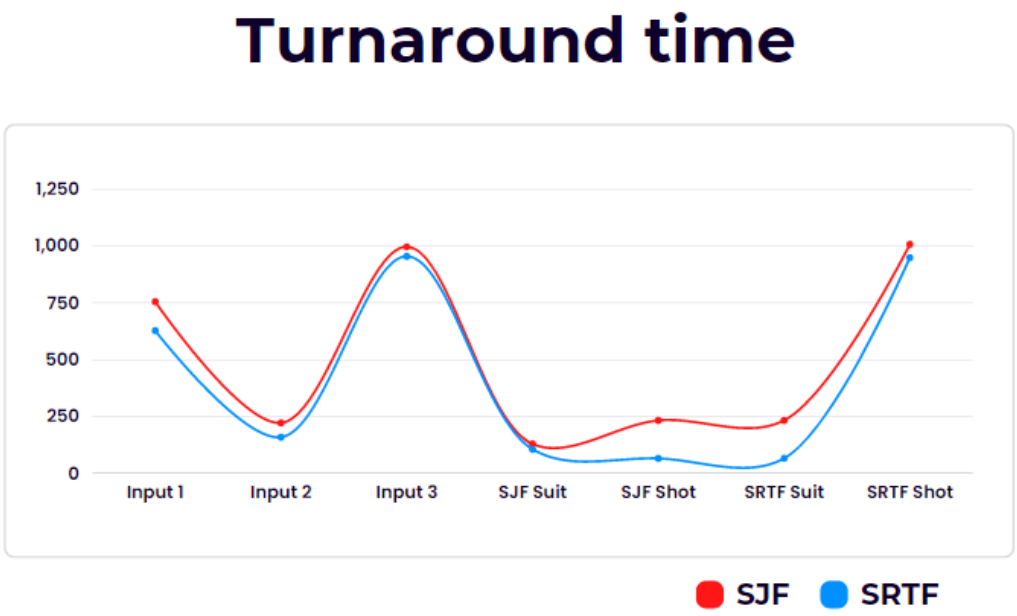

Figure 18: Turnaround time

For Turnaround time, we already know that a lower turnaround time implies that the scheduler has performed better. As we can see, SRTF was able to give lesser turnaround times in all cases because of its pre-emptive nature.

3.2 Waiting Time

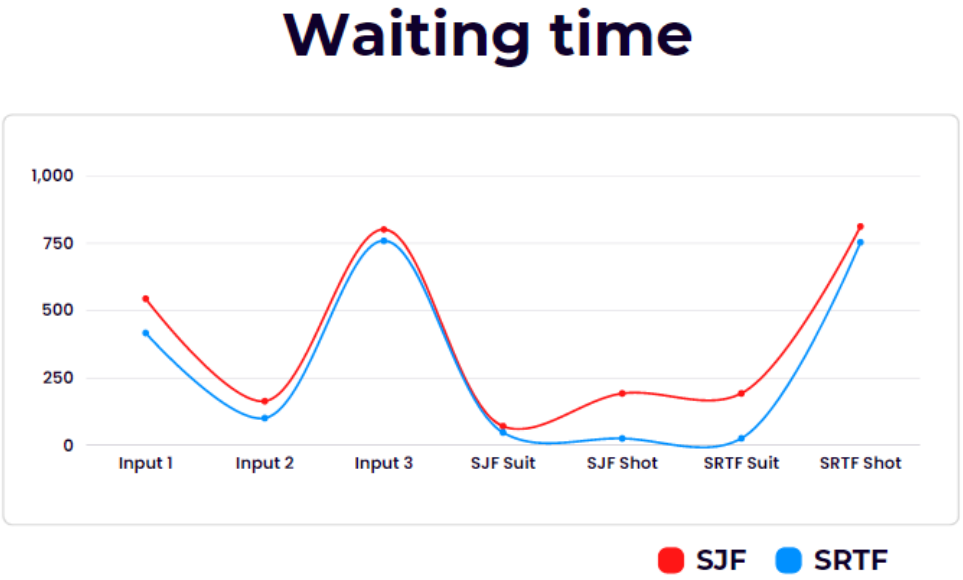

Figure 19: Waiting time

For Waiting time as well, a lower average waiting time throughout the dataset implies that the scheduler has performed better. Here, we can see that SRTF has a lower waiting time at all times, again due to its pre-emptive nature.

3.3 Penalty Ratio

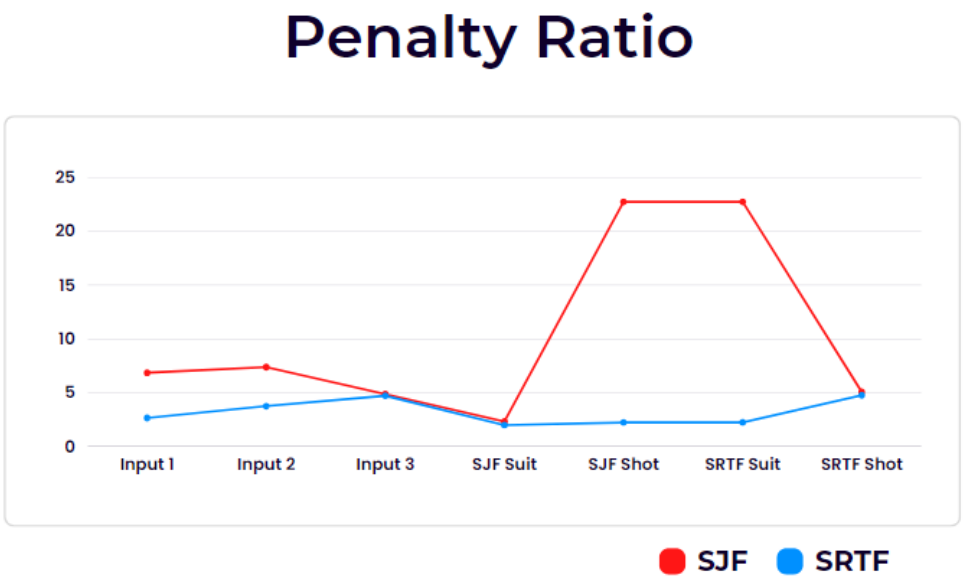

Figure 20: Penalty Ratio

A lower penalty ratio means that the scheduler has optimized the running of processes better. We can observe that there's a stark contrast between the penalty ratios of SRTF and SJF, especially in the case of SJF Shot and SRTF Suit where processes arrive at different times. SRTF performs especially well in this case since it swaps in and out processes at different time slices as specified by the user.

4 Conclusion

We have simulated the functionalities of SJF and SRTF schedulers over different datasets and observed their performances. While we can say definitively that SRTF has a better chance at performance than SJF at most cases, SRTF has its own vulnerabilities as well. For instance, SRTF does not solve the problem of CPU starvation for processes. Therefore, we adopt different scheduling mechanisms.

A bit of additional trivia would be that at a significantly high time slice value, SRTF would lose its capabilities of pre-emption and would change into an SJF scheduler.