Kuwait University

College of Engineering and Petroleum

Computer Engineering Department

CpE-445: OPERATING SYSTEM PRINCIPLES

Semester: SPRING / 2024

Section No. 03A

Assignment No. Project

Asmaa Adel Alazmi	2201122708
Reem Bader Alshammari	2211139357

Instructor Name: Dr. Lulwah Alhubail

TA Name: Eng. Abeer Alfadhel

Date: 5/5/2024

Table of Contents

Introduction

CPU scheduling is about choosing which process gets to use the CPU for running its tasks, while other processes wait. It's important because it helps use resources efficiently and improves how well the system works. The main aim of CPU scheduling is to keep the CPU busy and to reduce system overhead. The operating system ensures there's always a process ready to go, so the CPU always has something to do.

Methodology

The algorithms mentioned in this report will be programmed and tested using the programming language Python (version 3.11.0 and above). With the support of the API library Streamlit, an interface for the results will be provided. The test will be done by running two input files for each algorithm, then for the same inputs a graph will be constructed to compare the performance of all the algorithms. The performance will be judged based on the average of response time, waiting time and turnaround time. The input files are assumed to follow the following text file structure, with the first line being the quantum then following it are the lines for the process's information. The first column will be the process ID, second column will be the arrival time, third column will be the burst time and the last column will be the process priority.

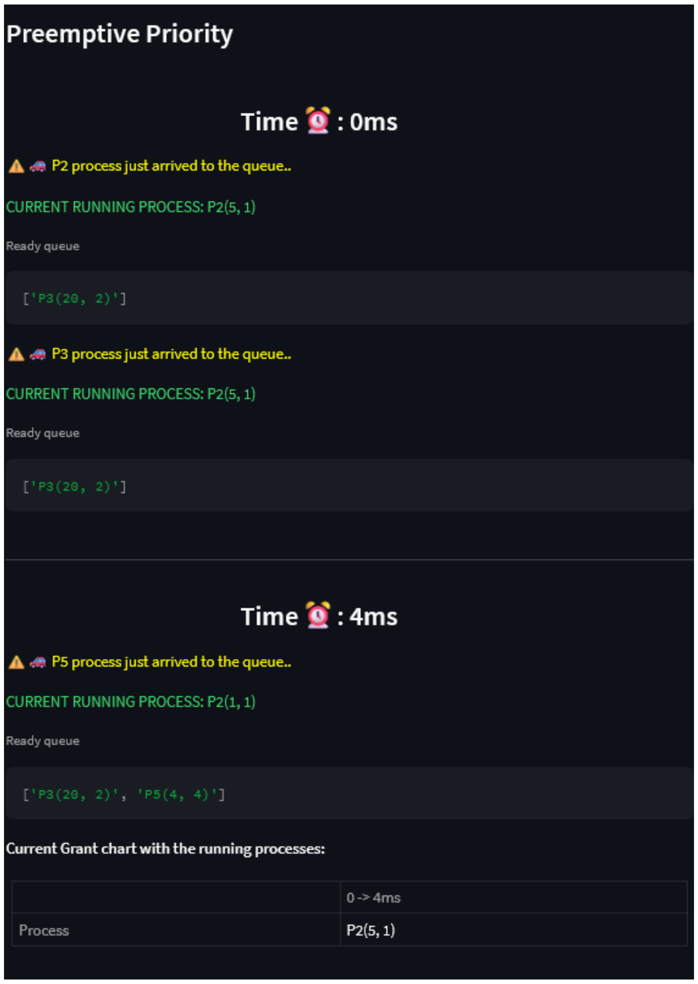
First input file	Second input file

4
1 6 5 1
2 0 5 1
3 0 20 2
4 10 10 3
5 4 4 4
	6
	1 0 5 1
	2 2 10 3
	3 10 6 2
	4 5 6 1

i. Preemptive Priority (PP)

Preemptive Priority (PP) is a type of priority scheduling algorithm. All processes carry 3 main values within them, arrive time, burst time and in this case priority number. Each process is assigned a priority that indicates the urgency of the process. The smallest the priority number is the highest the priority. In the case of this algorithm, the scheduling is preemptive meaning that if a new process arrives to the CPU ready queue with a higher priority than the currently running process a context switch will happen and the new process will be running in the CPU after moving the previous running process to the waiting queue.

The Output of the PP algorithm for the first input file:



Preemptive Priority

Time ⏰ : 0ms

⚠️ 🚙 P2 process just arrived to the queue..

CURRENT RUNNING PROCESS: P2(5, 1)

Ready queue

```
['P3(20, 2)']
```

⚠️ 🚙 P3 process just arrived to the queue..

CURRENT RUNNING PROCESS: P2(5, 1)

Ready queue

```
['P3(20, 2)']
```

Time ⏰ : 4ms

⚠️ 🚙 P5 process just arrived to the queue..

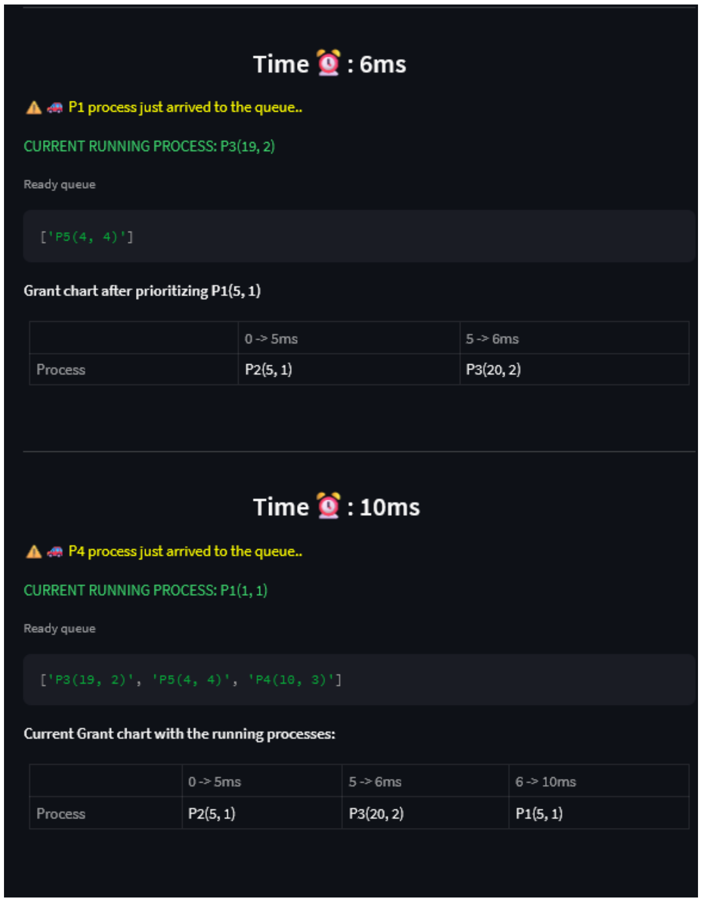
CURRENT RUNNING PROCESS: P2(1, 1)

Ready queue

```
['P3(20, 2)', 'P5(4, 4)']
```

Current Grant chart with the running processes:

	0 -> 4ms
Process	P2(5, 1)


Time ⏰ : 6ms

⚠️ 🚙 P1 process just arrived to the queue..

CURRENT RUNNING PROCESS: P3(19, 2)

Ready queue

```
['P5(4, 4)']
```

Grant chart after prioritizing P1(5, 1)

	0 -> 5ms	5 -> 6ms
Process	P2(5, 1)	P3(20, 2)

Time ⏰ : 10ms

⚠️ 🚙 P4 process just arrived to the queue..

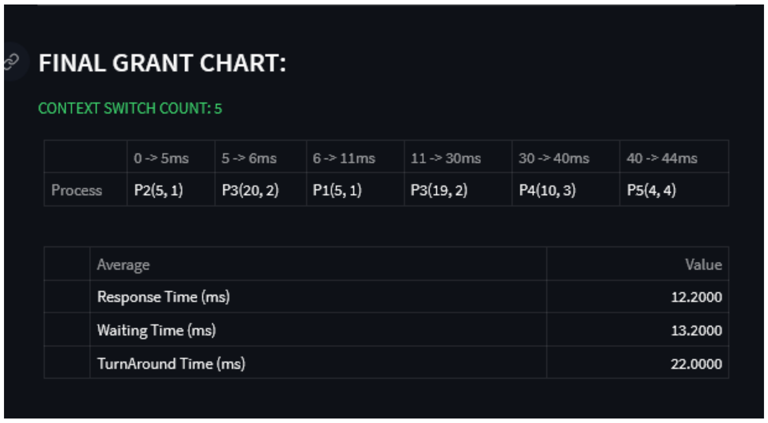
CURRENT RUNNING PROCESS: P1(1, 1)

Ready queue

```
['P3(19, 2)', 'P5(4, 4)', 'P4(10, 3)']
```

Current Grant chart with the running processes:

	0 -> 5ms	5 -> 6ms	6 -> 10ms
Process	P2(5, 1)	P3(20, 2)	P1(5, 1)

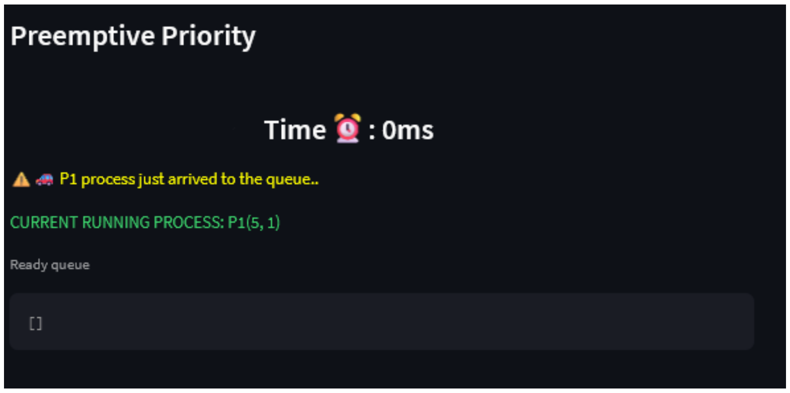

FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 5

	0 -> 5ms	5 -> 6ms	6 -> 11ms	11 -> 30ms	30 -> 40ms	40 -> 44ms
Process	P2(5, 1)	P3(20, 2)	P1(5, 1)	P3(19, 2)	P4(10, 3)	P5(4, 4)

	Average	Value
	Response Time (ms)	12.2000
	Waiting Time (ms)	13.2000
	TurnAround Time (ms)	22.0000

The Output of the PP algorithm for the second input file:



Preemptive Priority

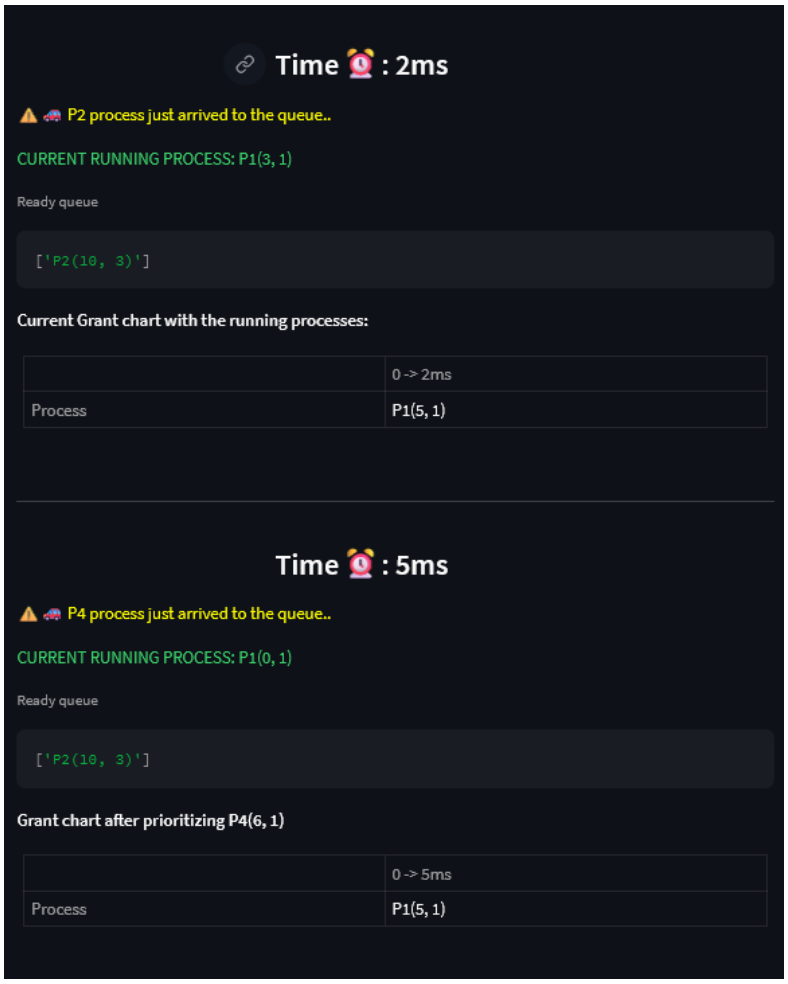
Time ⏰ : 0ms

⚠ 🚗 P1 process just arrived to the queue..

CURRENT RUNNING PROCESS: P1(5, 1)

Ready queue

```
[]
```



Time ⏰ : 2ms

⚠ 🚗 P2 process just arrived to the queue..

CURRENT RUNNING PROCESS: P1(3, 1)

Ready queue

```
['P2(10, 3)']
```

Current Grant chart with the running processes:

	0 -> 2ms
Process	P1(5, 1)

Time ⏰ : 5ms

⚠ 🚗 P4 process just arrived to the queue..

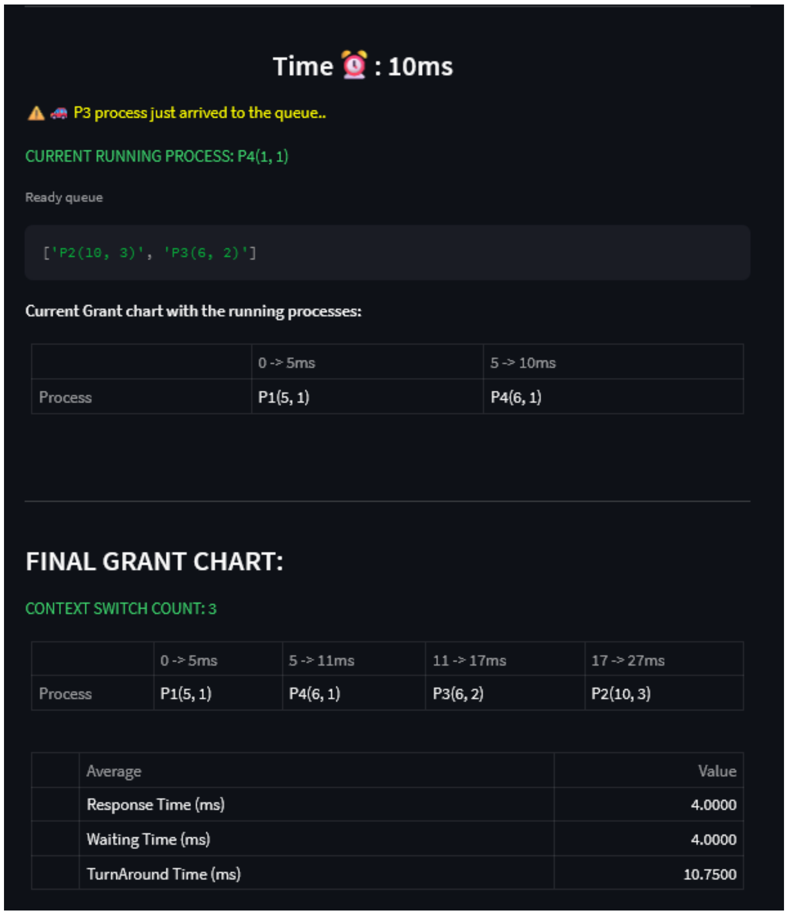
CURRENT RUNNING PROCESS: P1(0, 1)

Ready queue

```
['P2(10, 3)']
```

Grant chart after prioritizing P4(6, 1)

	0 -> 5ms
Process	P1(5, 1)


Time ⏰ : 10ms

⚠️ 🚓 P3 process just arrived to the queue..

CURRENT RUNNING PROCESS: P4(1, 1)

Ready queue

```
['P2(10, 3)', 'P3(6, 2)']
```

Current Grant chart with the running processes:

	0 -> 5ms	5 -> 10ms
Process	P1(5, 1)	P4(6, 1)

FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 3

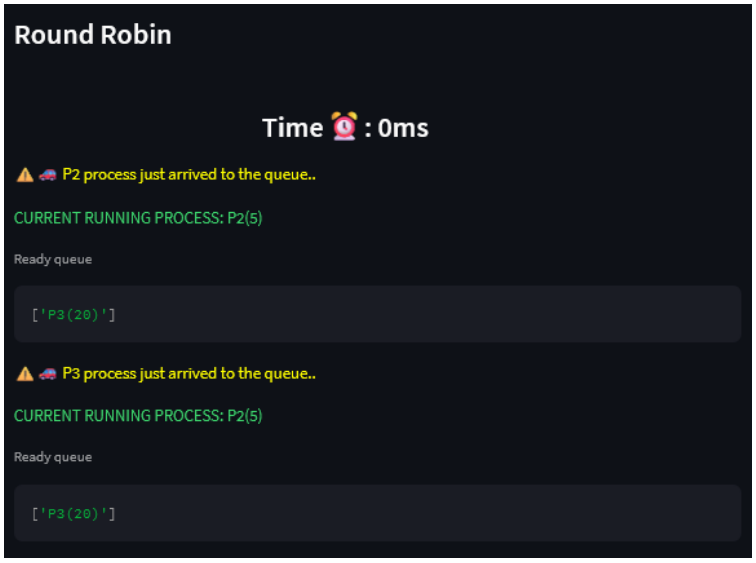
	0 -> 5ms	5 -> 11ms	11 -> 17ms	17 -> 27ms
Process	P1(5, 1)	P4(6, 1)	P3(6, 2)	P2(10, 3)

	Average	Value
	Response Time (ms)	4.0000
	Waiting Time (ms)	4.0000
	TurnAround Time (ms)	10.7500

ii. Round Robin (RR)

Round Robin is a fundamental scheduling technique used by operating systems to distribute CPU time among processes. It operates by allocating each process a fixed time slice, known as a "quantum," during which it can execute. Once the quantum expires, the CPU switches to the next process in line, ensuring fair treatment for all tasks.

The Output of the RR algorithm for the first input file:



Round Robin

Time ⏰ : 0ms

⚠️ 🚙 P2 process just arrived to the queue..

CURRENT RUNNING PROCESS: P2(5)

Ready queue

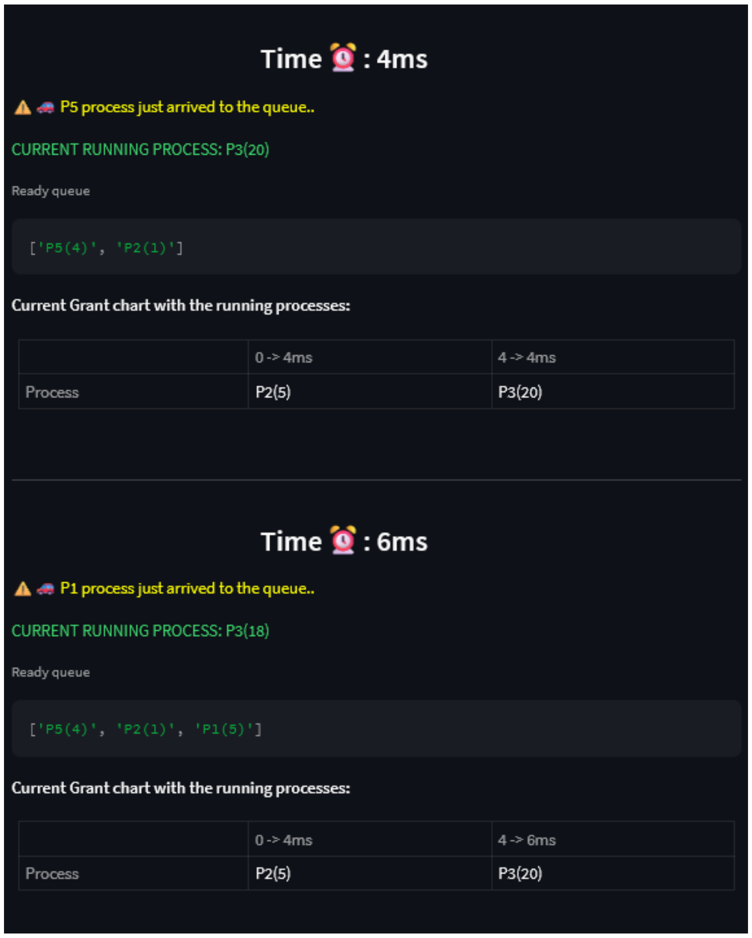
```
['P3(20)']
```

⚠️ 🚙 P3 process just arrived to the queue..

CURRENT RUNNING PROCESS: P2(5)

Ready queue

```
['P3(20)']
```



Time ⏰ : 4ms

⚠️ 🚙 P5 process just arrived to the queue..

CURRENT RUNNING PROCESS: P3(20)

Ready queue

```
['P5(4)', 'P2(1)']
```

Current Grant chart with the running processes:

	0 -> 4ms	4 -> 4ms
Process	P2(5)	P3(20)

Time ⏰ : 6ms

⚠️ 🚙 P1 process just arrived to the queue..

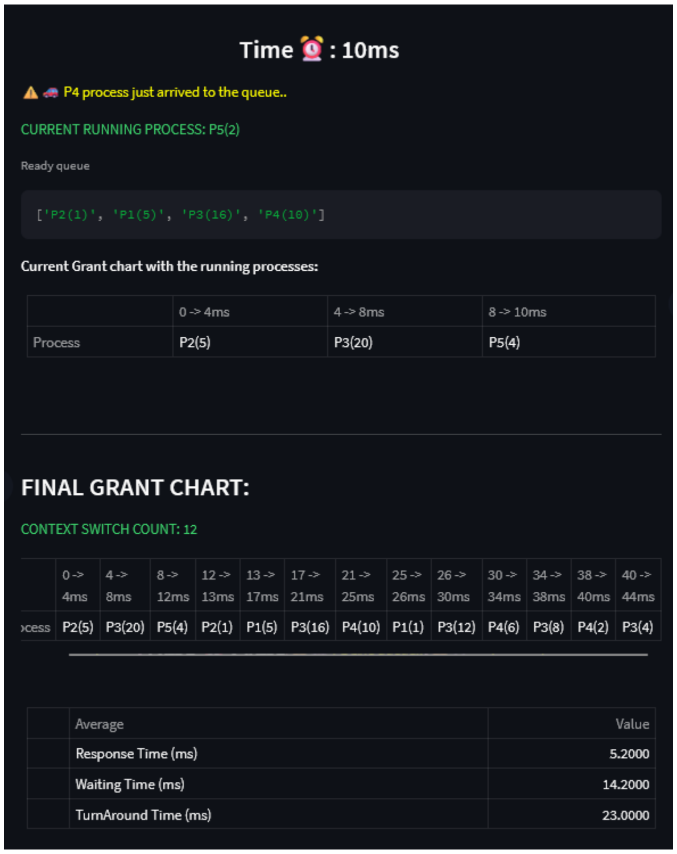
CURRENT RUNNING PROCESS: P3(18)

Ready queue

```
['P5(4)', 'P2(1)', 'P1(5)']
```

Current Grant chart with the running processes:

	0 -> 4ms	4 -> 6ms
Process	P2(5)	P3(20)



Time ⏰ : 10ms

⚠️ 🚗 P4 process just arrived to the queue..

CURRENT RUNNING PROCESS: P5(2)

Ready queue

```
['P2(1)', 'P1(5)', 'P3(16)', 'P4(10)']
```

Current Grant chart with the running processes:

	0 -> 4ms	4 -> 8ms	8 -> 10ms
Process	P2(5)	P3(20)	P5(4)

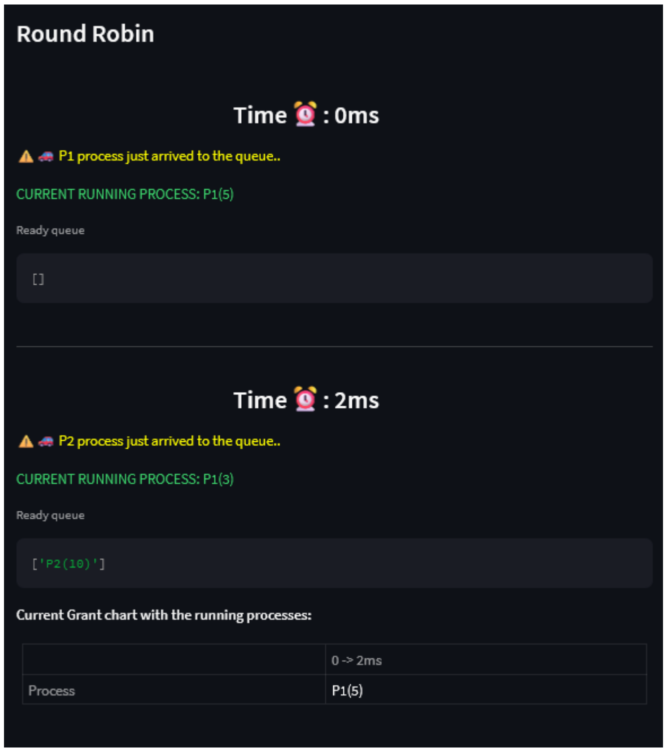
FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 12

	0 -> 4ms	4 -> 8ms	8 -> 12ms	12 -> 13ms	13 -> 17ms	17 -> 21ms	21 -> 25ms	25 -> 26ms	26 -> 30ms	30 -> 34ms	34 -> 38ms	38 -> 40ms	40 -> 44ms
ocess	P2(5)	P3(20)	P5(4)	P2(1)	P1(5)	P3(16)	P4(10)	P1(1)	P3(12)	P4(6)	P3(8)	P4(2)	P3(4)

	Average	Value
	Response Time (ms)	5.2000
	Waiting Time (ms)	14.2000
	TurnAround Time (ms)	23.0000

The Output of the RR algorithm for the second input file:



Round Robin

Time ⏰ : 0ms

⚠️ 🚗 P1 process just arrived to the queue..

CURRENT RUNNING PROCESS: P1(5)

Ready queue

```
[]
```

Time ⏰ : 2ms

⚠️ 🚗 P2 process just arrived to the queue..

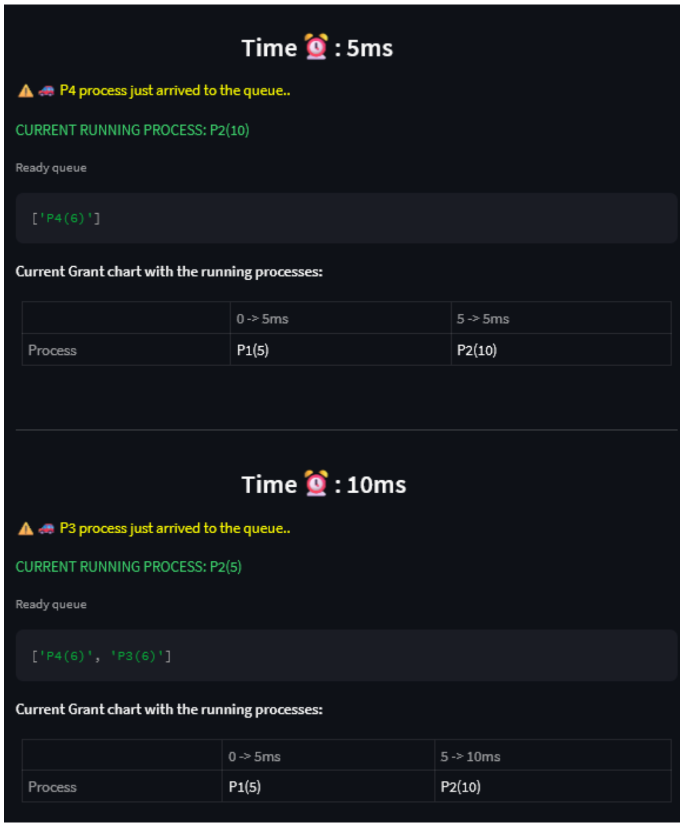
CURRENT RUNNING PROCESS: P1(3)

Ready queue

```
['P2(10)']
```

Current Grant chart with the running processes:

	0 -> 2ms
Process	P1(5)


Time ⏰ : 5ms

⚠ 🚙 P4 process just arrived to the queue..

CURRENT RUNNING PROCESS: P2(10)

Ready queue

```
['P4(6)']
```

Current Grant chart with the running processes:

	0 -> 5ms	5 -> 5ms
Process	P1(5)	P2(10)

Time ⏰ : 10ms

⚠ 🚙 P3 process just arrived to the queue..

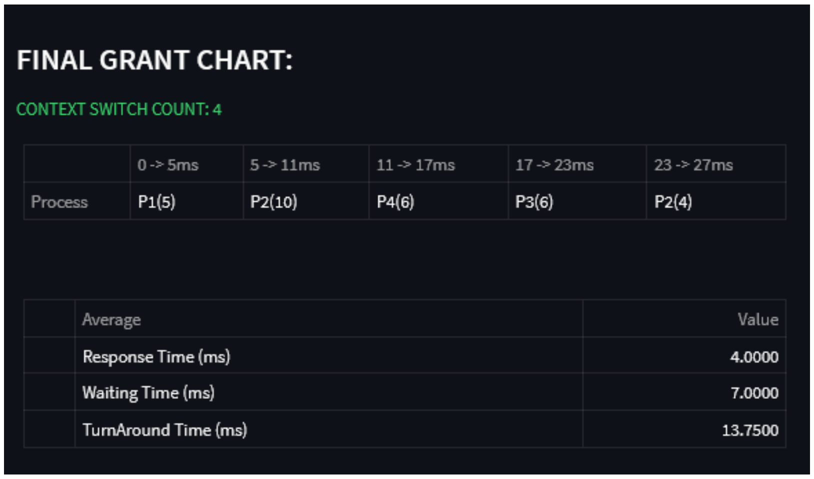
CURRENT RUNNING PROCESS: P2(5)

Ready queue

```
['P4(6)', 'P3(6)']
```

Current Grant chart with the running processes:

	0 -> 5ms	5 -> 10ms
Process	P1(5)	P2(10)


FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 4

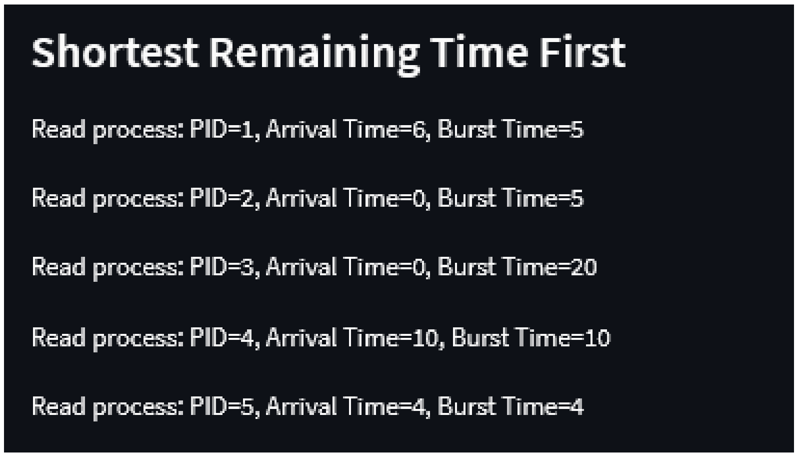
	0 -> 5ms	5 -> 11ms	11 -> 17ms	17 -> 23ms	23 -> 27ms
Process	P1(5)	P2(10)	P4(6)	P3(6)	P2(4)

	Average	Value
	Response Time (ms)	4.0000
	Waiting Time (ms)	7.0000
	TurnAround Time (ms)	13.7500

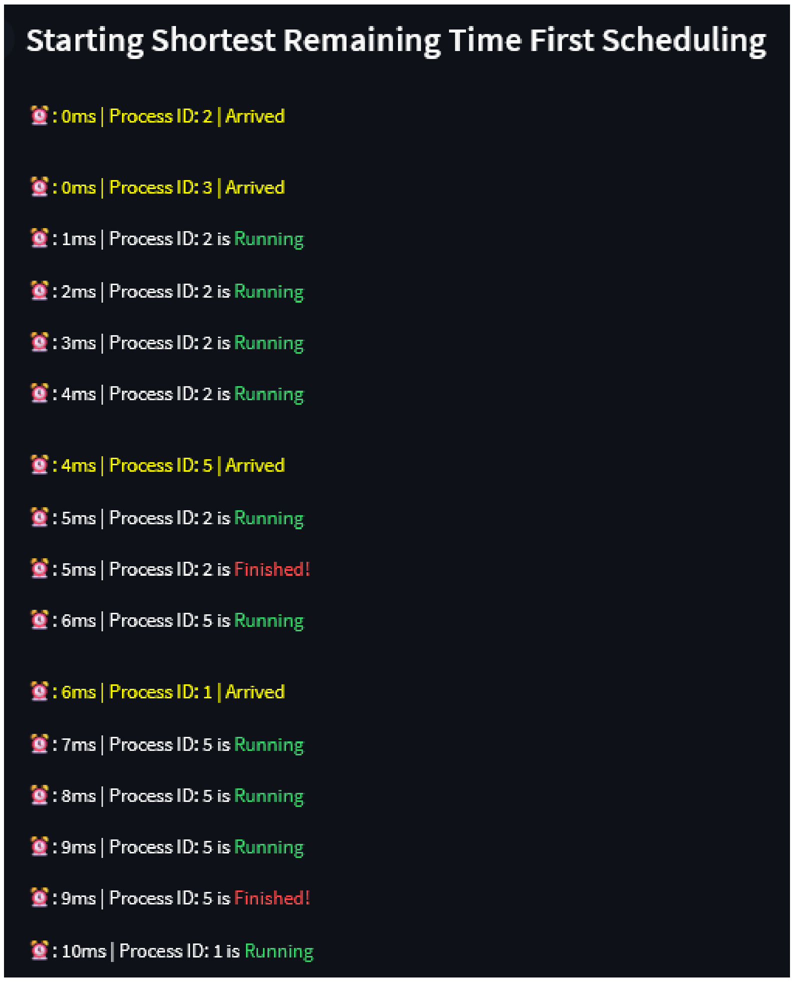
iii. Shortest Remaining Time First (SRTF)

Shortest Remaining Time First (SRTF), is an algorithm that prioritizes tasks based on their remaining duration. It ensures that the task with the shortest remaining time runs first until completion. Meanwhile, the scheduler maintains a queue of other processes awaiting execution. For this algorithm another printing format was used, the system will print the current state every milli second.

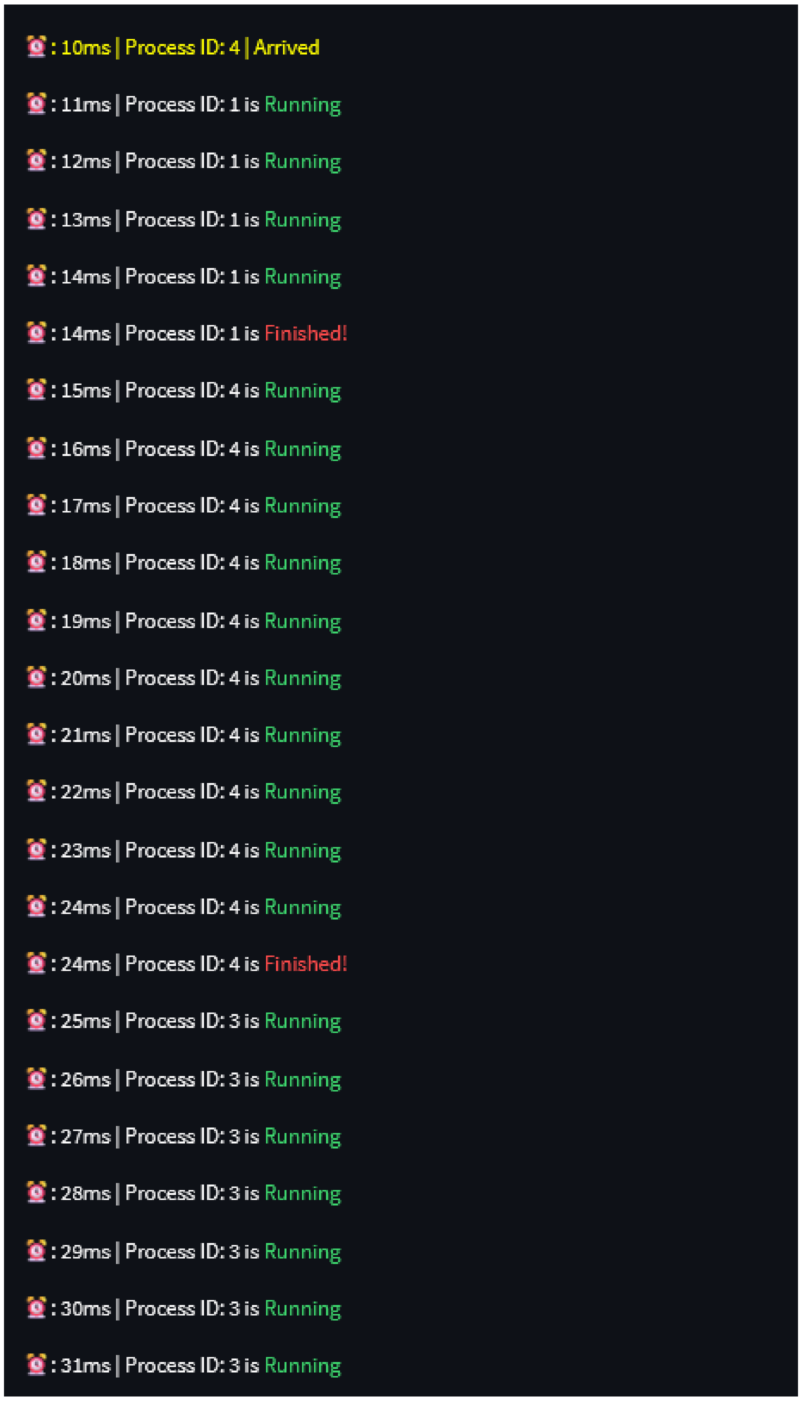
The Output of the RR algorithm for the first input file:



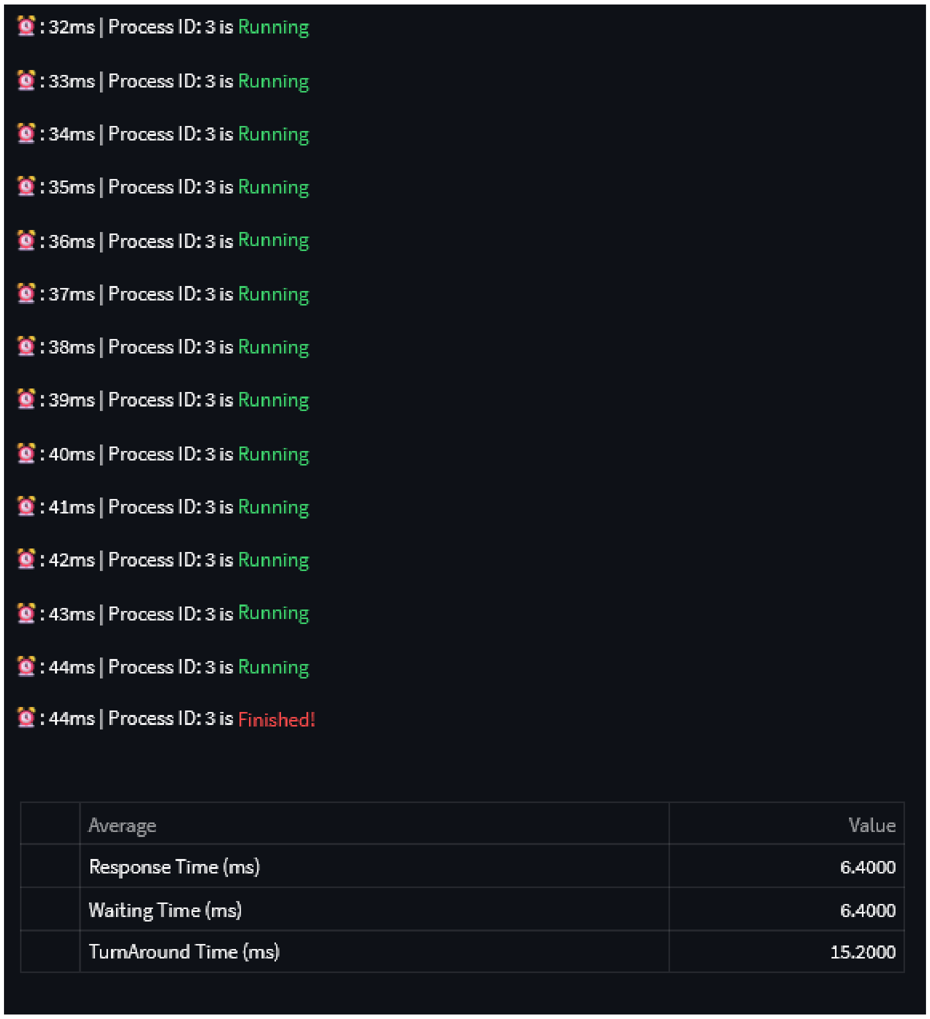
```
Shortest Remaining Time First

Read process: PID=1, Arrival Time=6, Burst Time=5

Read process: PID=2, Arrival Time=0, Burst Time=5

Read process: PID=3, Arrival Time=0, Burst Time=20

Read process: PID=4, Arrival Time=10, Burst Time=10

Read process: PID=5, Arrival Time=4, Burst Time=4
```



```
Starting Shortest Remaining Time First Scheduling

⏰ : 0ms | Process ID: 2 | Arrived

⏰ : 0ms | Process ID: 3 | Arrived
⏰ : 1ms | Process ID: 2 is Running
⏰ : 2ms | Process ID: 2 is Running
⏰ : 3ms | Process ID: 2 is Running
⏰ : 4ms | Process ID: 2 is Running

⏰ : 4ms | Process ID: 5 | Arrived
⏰ : 5ms | Process ID: 2 is Running
⏰ : 5ms | Process ID: 2 is Finished!
⏰ : 6ms | Process ID: 5 is Running

⏰ : 6ms | Process ID: 1 | Arrived
⏰ : 7ms | Process ID: 5 is Running
⏰ : 8ms | Process ID: 5 is Running
⏰ : 9ms | Process ID: 5 is Running
⏰ : 9ms | Process ID: 5 is Finished!
⏰ : 10ms | Process ID: 1 is Running
```



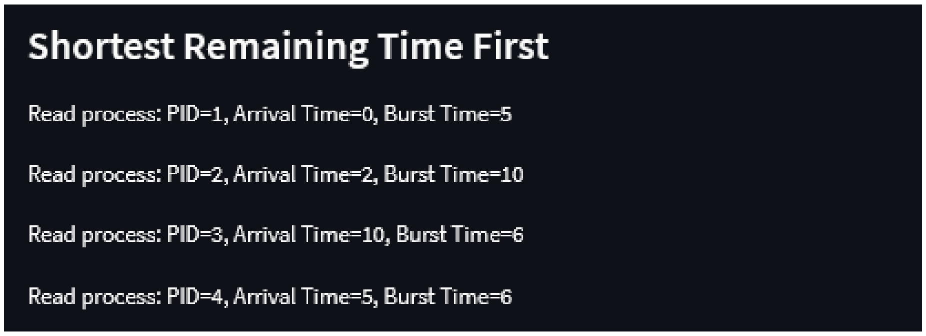
```
⏰: 10ms | Process ID: 4 | Arrived
⏰: 11ms | Process ID: 1 is Running
⏰: 12ms | Process ID: 1 is Running
⏰: 13ms | Process ID: 1 is Running
⏰: 14ms | Process ID: 1 is Running
⏰: 14ms | Process ID: 1 is Finished!
⏰: 15ms | Process ID: 4 is Running
⏰: 16ms | Process ID: 4 is Running
⏰: 17ms | Process ID: 4 is Running
⏰: 18ms | Process ID: 4 is Running
⏰: 19ms | Process ID: 4 is Running
⏰: 20ms | Process ID: 4 is Running
⏰: 21ms | Process ID: 4 is Running
⏰: 22ms | Process ID: 4 is Running
⏰: 23ms | Process ID: 4 is Running
⏰: 24ms | Process ID: 4 is Running
⏰: 24ms | Process ID: 4 is Finished!
⏰: 25ms | Process ID: 3 is Running
⏰: 26ms | Process ID: 3 is Running
⏰: 27ms | Process ID: 3 is Running
⏰: 28ms | Process ID: 3 is Running
⏰: 29ms | Process ID: 3 is Running
⏰: 30ms | Process ID: 3 is Running
⏰: 31ms | Process ID: 3 is Running
```



```
⏰ : 32ms | Process ID: 3 is Running
⏰ : 33ms | Process ID: 3 is Running
⏰ : 34ms | Process ID: 3 is Running
⏰ : 35ms | Process ID: 3 is Running
⏰ : 36ms | Process ID: 3 is Running
⏰ : 37ms | Process ID: 3 is Running
⏰ : 38ms | Process ID: 3 is Running
⏰ : 39ms | Process ID: 3 is Running
⏰ : 40ms | Process ID: 3 is Running
⏰ : 41ms | Process ID: 3 is Running
⏰ : 42ms | Process ID: 3 is Running
⏰ : 43ms | Process ID: 3 is Running
⏰ : 44ms | Process ID: 3 is Running
⏰ : 44ms | Process ID: 3 is Finished!
```

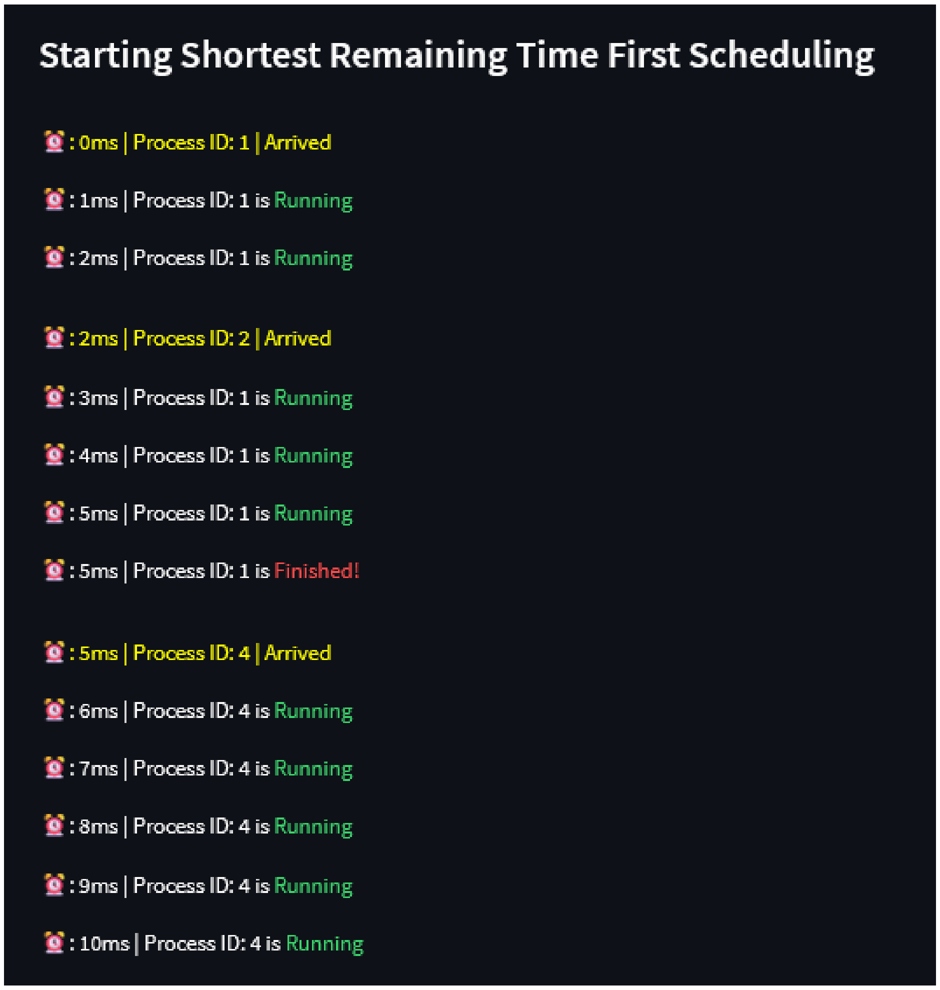
	Average	Value
	Response Time (ms)	6.4000
	Waiting Time (ms)	6.4000
	TurnAround Time (ms)	15.2000

The Output of the RR algorithm for the second input file:

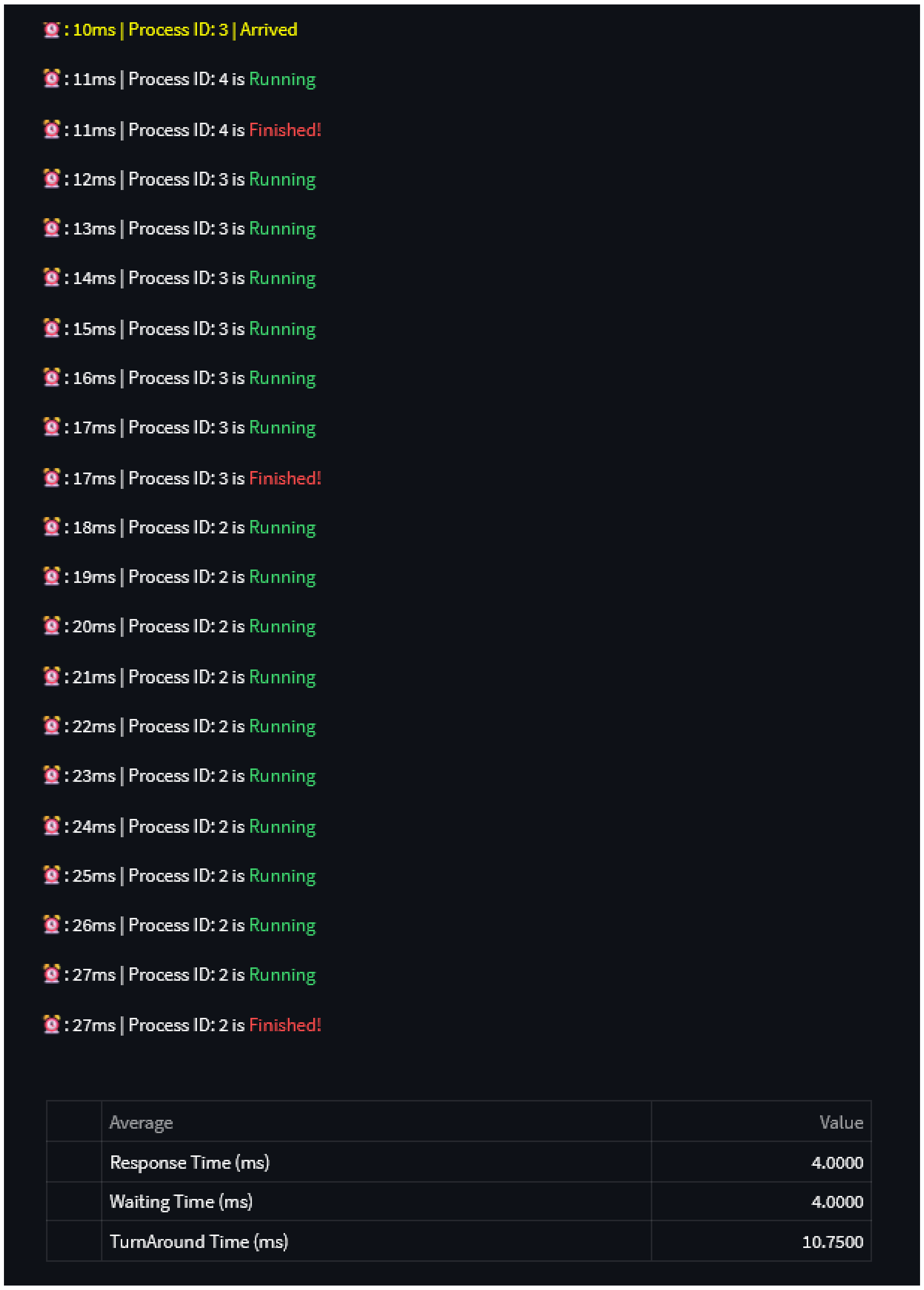


Shortest Remaining Time First

```
Read process: PID=1, Arrival Time=0, Burst Time=5
Read process: PID=2, Arrival Time=2, Burst Time=10
Read process: PID=3, Arrival Time=10, Burst Time=6
Read process: PID=4, Arrival Time=5, Burst Time=6
```



```
Starting Shortest Remaining Time First Scheduling

⏰ : 0ms | Process ID: 1 | Arrived

⏰ : 1ms | Process ID: 1 is Running

⏰ : 2ms | Process ID: 1 is Running

⏰ : 2ms | Process ID: 2 | Arrived

⏰ : 3ms | Process ID: 1 is Running

⏰ : 4ms | Process ID: 1 is Running

⏰ : 5ms | Process ID: 1 is Running

⏰ : 5ms | Process ID: 1 is Finished!

⏰ : 5ms | Process ID: 4 | Arrived

⏰ : 6ms | Process ID: 4 is Running

⏰ : 7ms | Process ID: 4 is Running

⏰ : 8ms | Process ID: 4 is Running

⏰ : 9ms | Process ID: 4 is Running

⏰ : 10ms | Process ID: 4 is Running
```

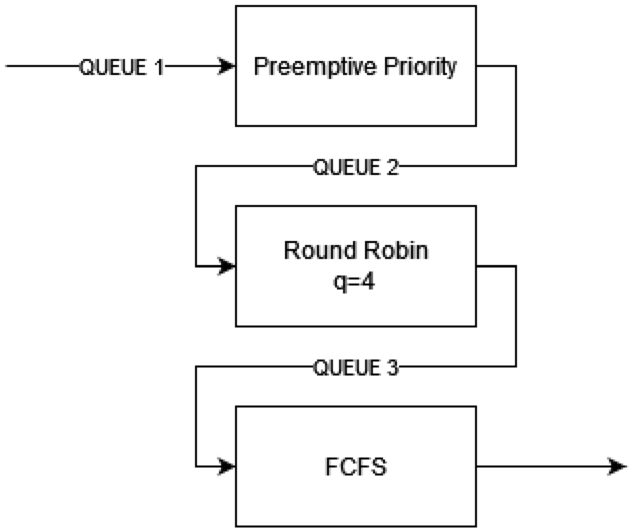


```
⏰ : 10ms | Process ID: 3 | Arrived
⏰ : 11ms | Process ID: 4 is Running
⏰ : 11ms | Process ID: 4 is Finished!
⏰ : 12ms | Process ID: 3 is Running
⏰ : 13ms | Process ID: 3 is Running
⏰ : 14ms | Process ID: 3 is Running
⏰ : 15ms | Process ID: 3 is Running
⏰ : 16ms | Process ID: 3 is Running
⏰ : 17ms | Process ID: 3 is Running
⏰ : 17ms | Process ID: 3 is Finished!
⏰ : 18ms | Process ID: 2 is Running
⏰ : 19ms | Process ID: 2 is Running
⏰ : 20ms | Process ID: 2 is Running
⏰ : 21ms | Process ID: 2 is Running
⏰ : 22ms | Process ID: 2 is Running
⏰ : 23ms | Process ID: 2 is Running
⏰ : 24ms | Process ID: 2 is Running
⏰ : 25ms | Process ID: 2 is Running
⏰ : 26ms | Process ID: 2 is Running
⏰ : 27ms | Process ID: 2 is Running
⏰ : 27ms | Process ID: 2 is Finished!
```

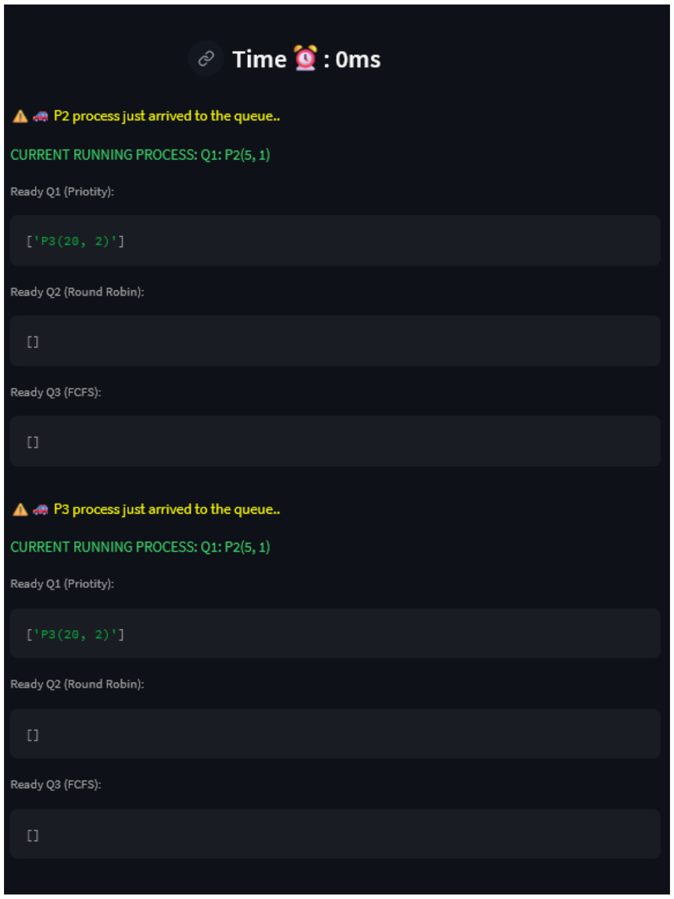
	Average	Value
	Response Time (ms)	4.0000
	Waiting Time (ms)	4.0000
	TurnAround Time (ms)	10.7500

iv. Multi-Level feedback queue Custom Algorithm

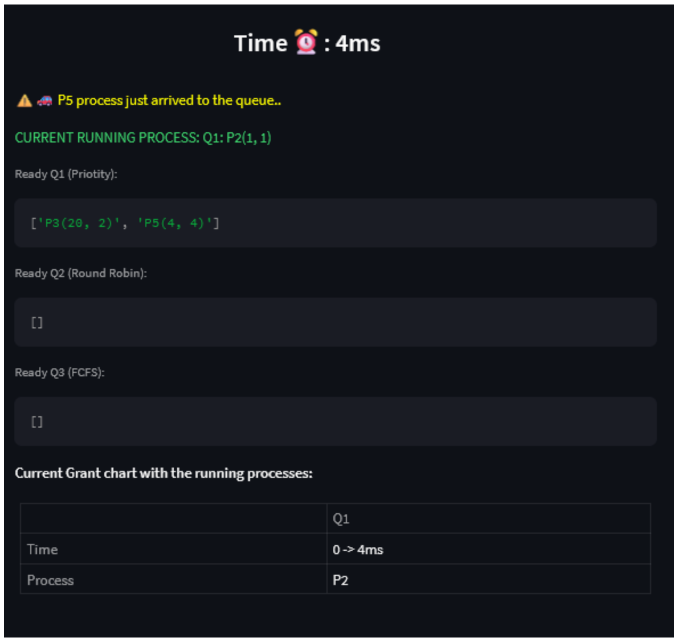
When the multilevel feedback queue scheduling algorithm is used, processes are able to move between queues. The idea is to separate the processes according to the characteristics of their CPU bursts. In some cases, a process might use too much CPU time and that leads other processes in the ready queue to suffer from starvation. That is why the MLFQ demotes such process to a lower priority queue, to help prevent this problem. For this custom MLFQ 3 queues were used, the queues have the demote functionality only (no promote). The following figure shows the scheduling algorithm used for each queue:



The Output of the MLFQ algorithm for the first input file:

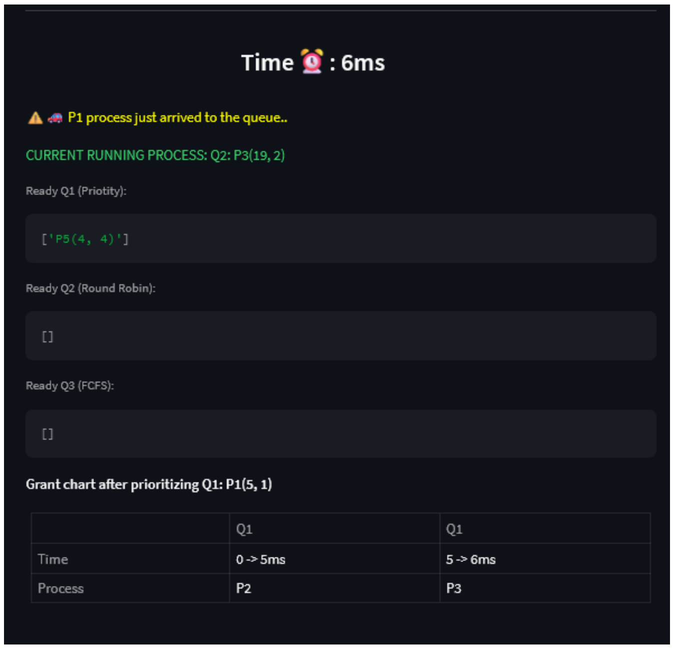


```
Time ⏰ : 0ms

⚠ 🚗 P2 process just arrived to the queue..
CURRENT RUNNING PROCESS: Q1: P2(5, 1)
Ready Q1 (Priotity):
['P3(20, 2)']
Ready Q2 (Round Robin):
[]
Ready Q3 (FCFS):
[]

⚠ 🚗 P3 process just arrived to the queue..
CURRENT RUNNING PROCESS: Q1: P2(5, 1)
Ready Q1 (Priotity):
['P3(20, 2)']
Ready Q2 (Round Robin):
[]
Ready Q3 (FCFS):
[]
```



```
Time ⏰ : 4ms

⚠ 🚗 P5 process just arrived to the queue..
CURRENT RUNNING PROCESS: Q1: P2(1, 1)
Ready Q1 (Priotity):
['P3(20, 2)', 'P5(4, 4)']
Ready Q2 (Round Robin):
[]
Ready Q3 (FCFS):
[]
Current Grant chart with the running processes:
```

	Q1
Time	0 -> 4ms
Process	P2



Time ⏰ : 6ms

⚠️ 🚓 P1 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q2: P3(19, 2)

Ready Q1 (Priotity):

```
['P5(4, 4)']
```

Ready Q2 (Round Robin):

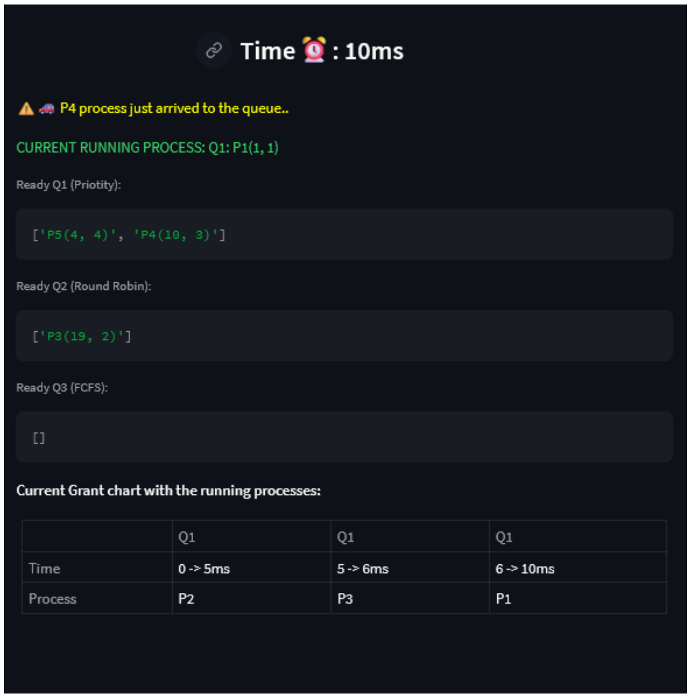
```
[]
```

Ready Q3 (FCFS):

```
[]
```

Grant chart after prioritizing Q1: P1(5, 1)

	Q1	Q1
Time	0 -> 5ms	5 -> 6ms
Process	P2	P3



Time ⏰ : 10ms

⚠️ 🚓 P4 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q1: P1(1, 1)

Ready Q1 (Priotity):

```
['P5(4, 4)', 'P4(10, 3)']
```

Ready Q2 (Round Robin):

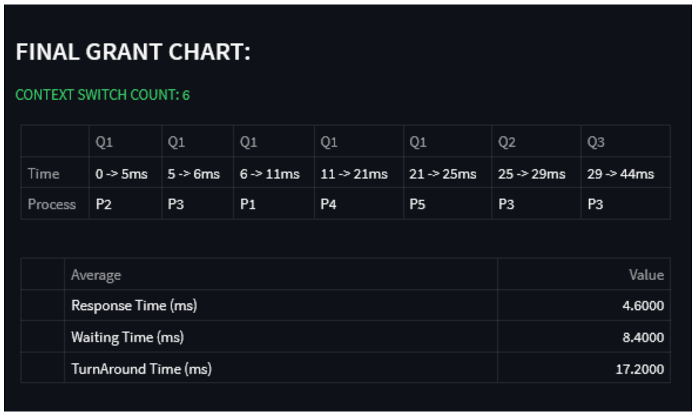
```
['P3(19, 2)']
```

Ready Q3 (FCFS):

```
[]
```

Current Grant chart with the running processes:

	Q1	Q1	Q1
Time	0 -> 5ms	5 -> 6ms	6 -> 10ms
Process	P2	P3	P1

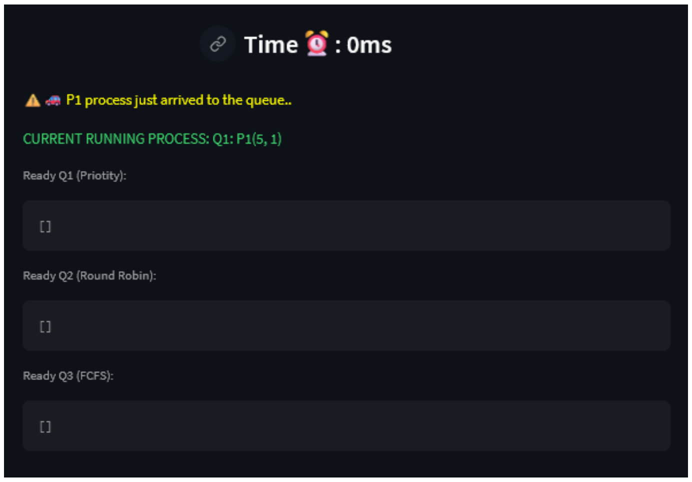

FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 6

	Q1	Q1	Q1	Q1	Q1	Q2	Q3
Time	0 -> 5ms	5 -> 6ms	6 -> 11ms	11 -> 21ms	21 -> 25ms	25 -> 29ms	29 -> 44ms
Process	P2	P3	P1	P4	P5	P3	P3

	Average	Value
	Response Time (ms)	4.6000
	Waiting Time (ms)	8.4000
	TurnAround Time (ms)	17.2000

The Output of the MLFQ algorithm for the second input file:


Time ⏰ : 0ms

⚠ 🚗 P1 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q1: P1(5, 1)

Ready Q1 (Priotity):

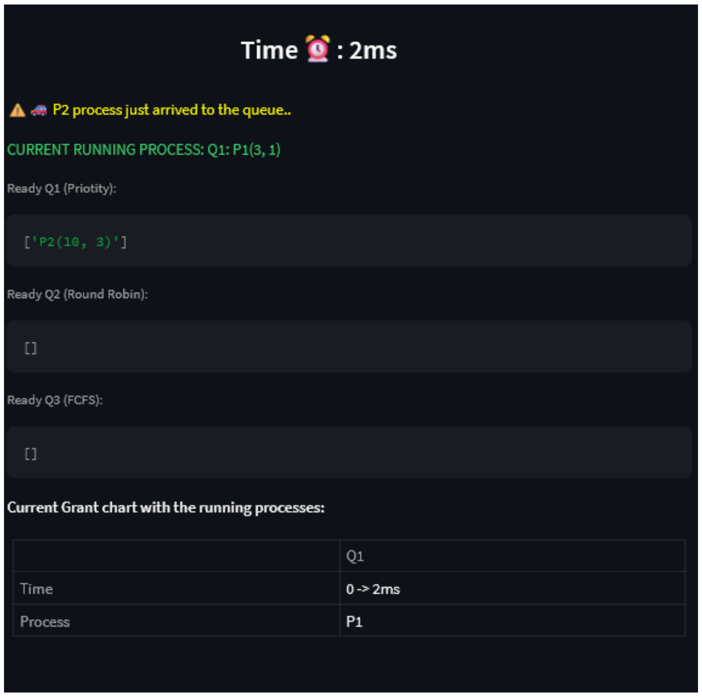
```
[]
```

Ready Q2 (Round Robin):

```
[]
```

Ready Q3 (FCFS):

```
[]
```


Time ⏰ : 2ms

⚠ 🚗 P2 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q1: P1(3, 1)

Ready Q1 (Priotity):

```
['P2(10, 3)']
```

Ready Q2 (Round Robin):

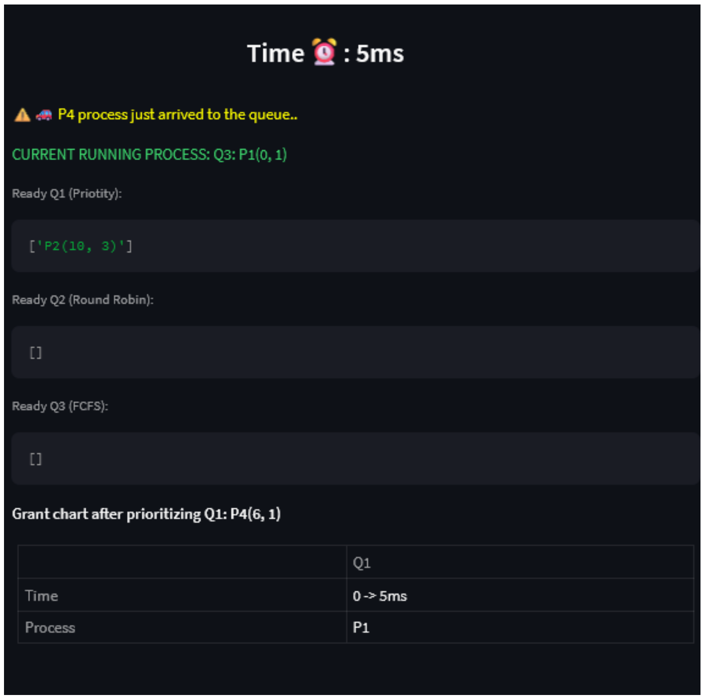
```
[]
```

Ready Q3 (FCFS):

```
[]
```

Current Grant chart with the running processes:

	Q1
Time	0 -> 2ms
Process	P1



Time ⏰ : 5ms

⚠ 🚓 P4 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q3: P1(0, 1)

Ready Q1 (Priotity):

```
['P2(10, 3)']
```

Ready Q2 (Round Robin):

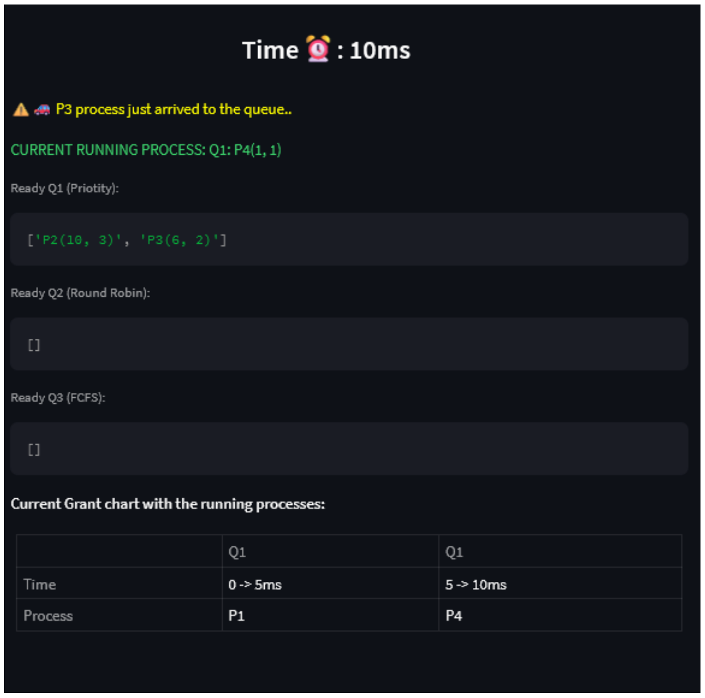
```
[]
```

Ready Q3 (FCFS):

```
[]
```

Grant chart after prioritizing Q1: P4(6, 1)

	Q1
Time	0 -> 5ms
Process	P1



Time ⏰ : 10ms

⚠ 🚓 P3 process just arrived to the queue..

CURRENT RUNNING PROCESS: Q1: P4(1, 1)

Ready Q1 (Priotity):

```
['P2(10, 3)', 'P3(6, 2)']
```

Ready Q2 (Round Robin):

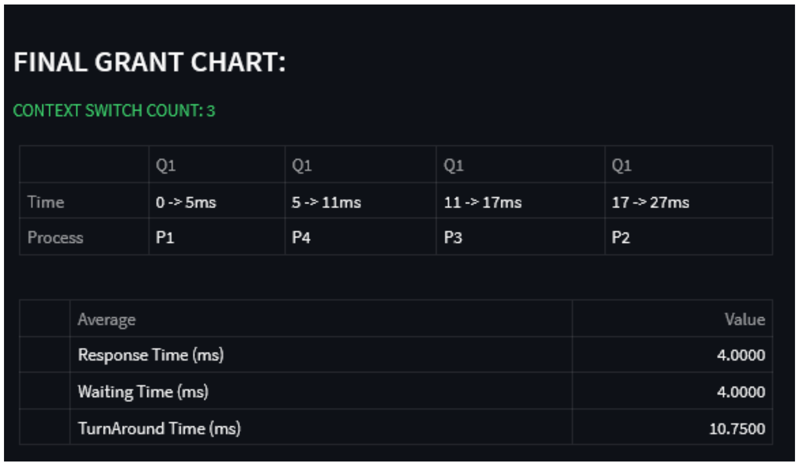
```
[]
```

Ready Q3 (FCFS):

```
[]
```

Current Grant chart with the running processes:

	Q1	Q1
Time	0 -> 5ms	5 -> 10ms
Process	P1	P4



FINAL GRANT CHART:

CONTEXT SWITCH COUNT: 3

	Q1	Q1	Q1	Q1
Time	0 -> 5ms	5 -> 11ms	11 -> 17ms	17 -> 27ms
Process	P1	P4	P3	P2

Average	Value
Response Time (ms)	4.0000
Waiting Time (ms)	4.0000
TurnAround Time (ms)	10.7500

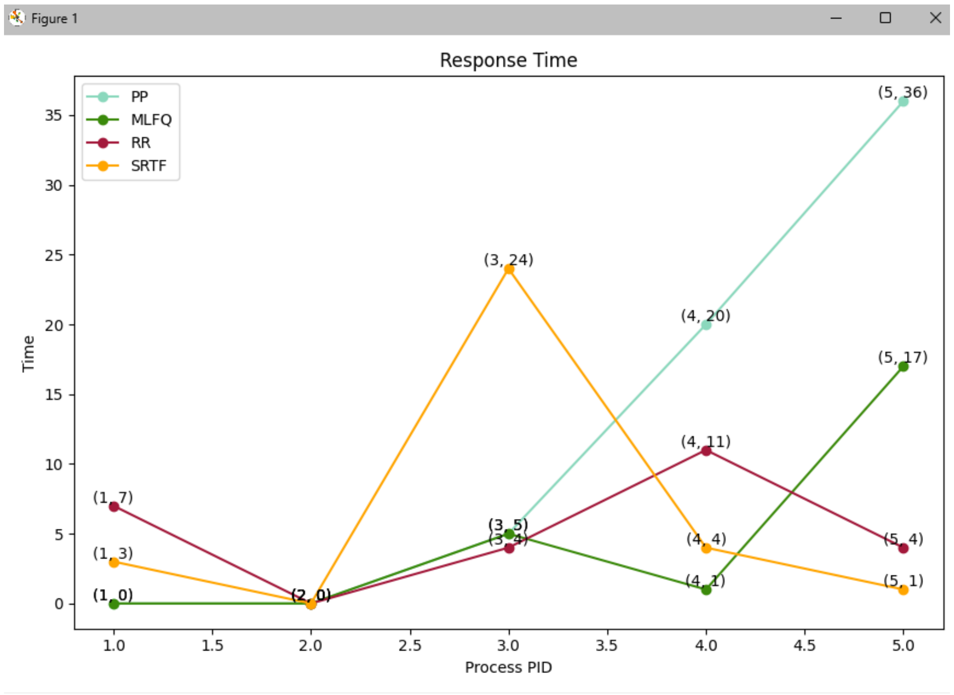
Results Analysis

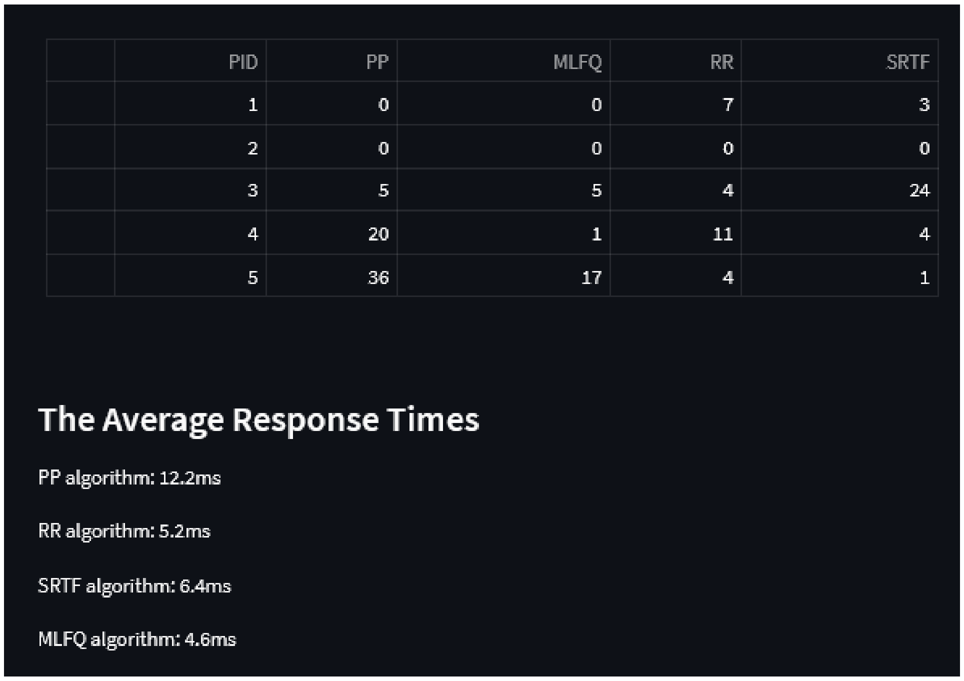
This section covers the analysis of the results obtained from the plotting graphs. These graphs and tables showcase the different recorded times for each scheduling algorithm. The comparison is made using the average response, waiting and turnaround times. The formula to calculate each of these times is as follow:

- **Response time (RT)**: the time stamp which the process first gets selected by the CPU to run.
- **Waiting time (WT)**: The cumulative duration for which the process has been waiting in the ready queue.
- **The turnaround time (TAT)**: the total duration the process spends in both the ready queue and the running state.

The plots were generated using the Python library Matplotlib. And the tables were generated using the Streamlit API and Pandas library, the values in the tables represent the time in milli seconds for each process in a certain scheduling algorithm.

(1) Plotting the first input file:





	PID	PP	MLFQ	RR	SRTF
	1	0	0	7	3
	2	0	0	0	0
	3	5	5	4	24
	4	20	1	11	4
	5	36	17	4	1

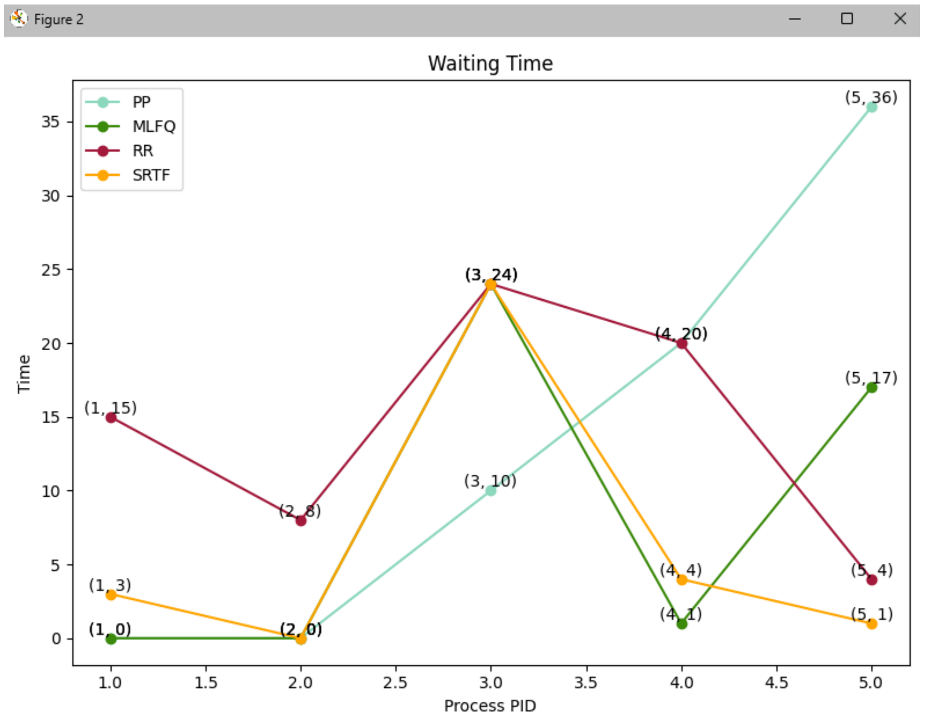
The Average Response Times

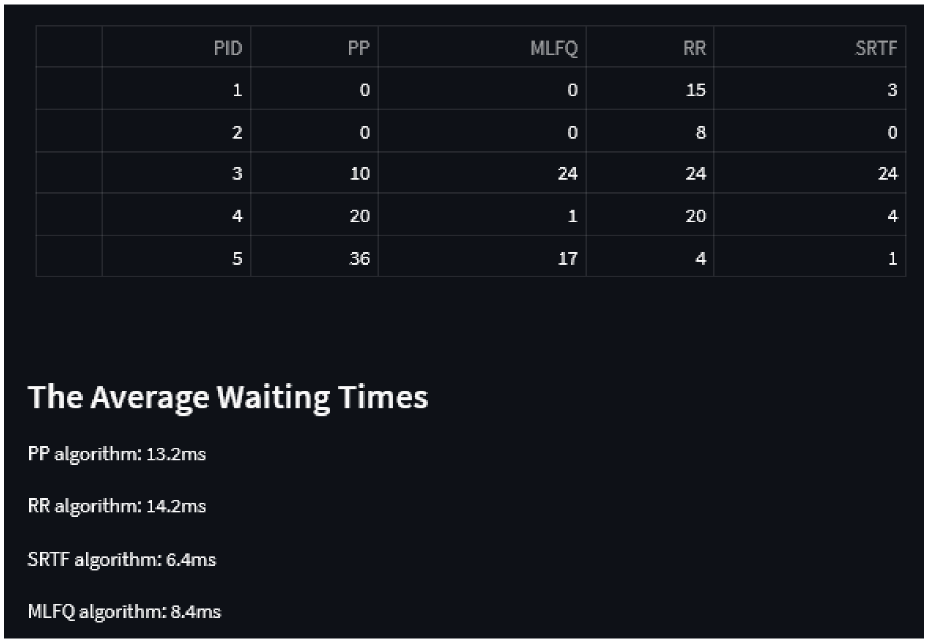
PP algorithm: 12.2ms

RR algorithm: 5.2ms

SRTF algorithm: 6.4ms

MLFQ algorithm: 4.6ms





	PID	PP	MLFQ	RR	SRTF
	1	0	0	15	3
	2	0	0	8	0
	3	10	24	24	24
	4	20	1	20	4
	5	36	17	4	1

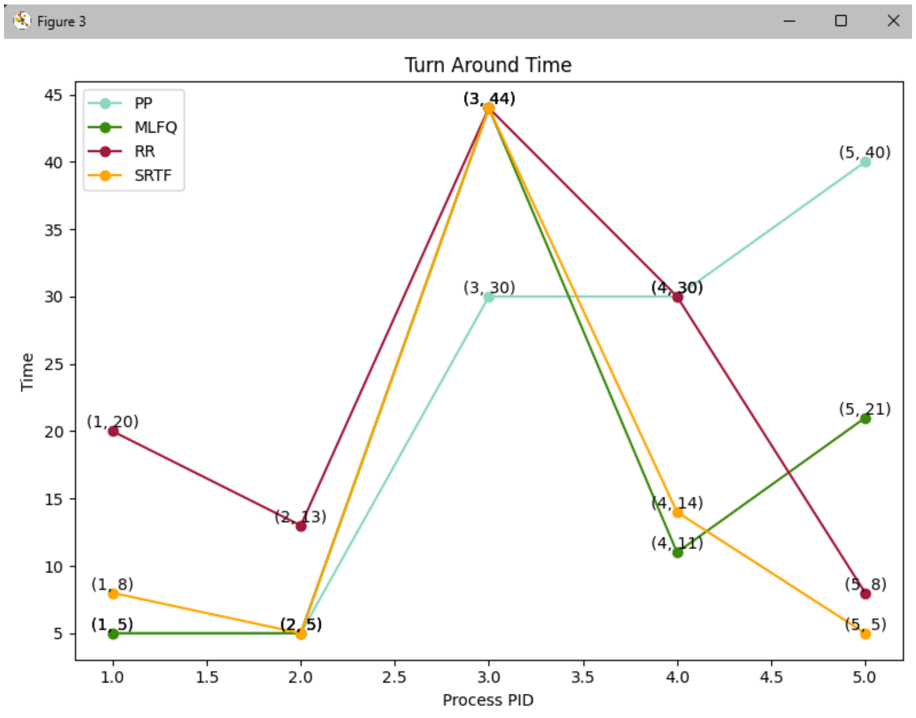
The Average Waiting Times

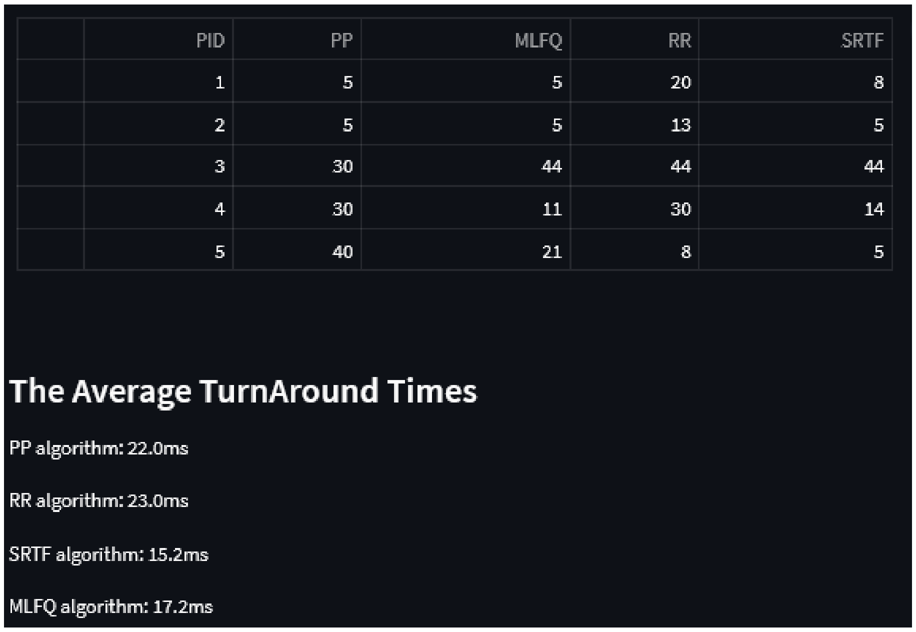
PP algorithm: 13.2ms

RR algorithm: 14.2ms

SRTF algorithm: 6.4ms

MLFQ algorithm: 8.4ms





	PID	PP	MLFQ	RR	SRTF
	1	5	5	20	8
	2	5	5	13	5
	3	30	44	44	44
	4	30	11	30	14
	5	40	21	8	5

The Average TurnAround Times

PP algorithm: 22.0ms

RR algorithm: 23.0ms

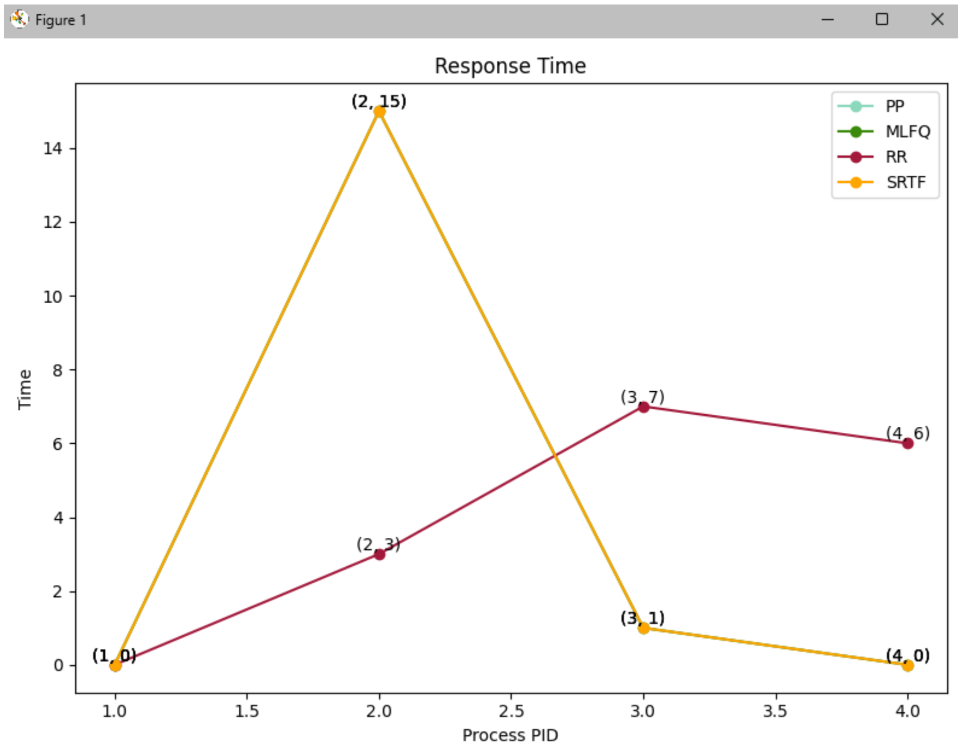
SRTF algorithm: 15.2ms

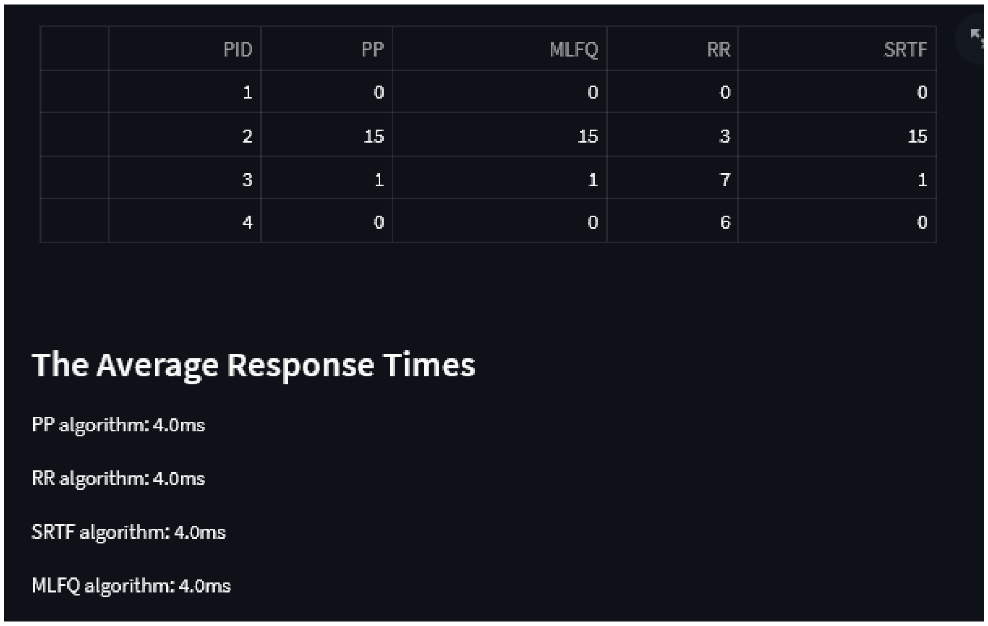
MLFQ algorithm: 17.2ms

The analysis of the results highly indicates the dominance of the SRTF algorithm. As seen, the average results for the SRTF algorithm ranks high in the TAT and WT criteria. Then comes next the MLFQ, using any of both these algorithms shall result in an overall better performance than the other two. The

SRTF might be the better option in this case, as the MLFQ tend to be more costly since it depends on three main queues rather than a single one.

(2) Plotting the second input file:





	PID	PP	MLFQ	RR	SRTF
	1	0	0	0	0
	2	15	15	3	15
	3	1	1	7	1
	4	0	0	6	0

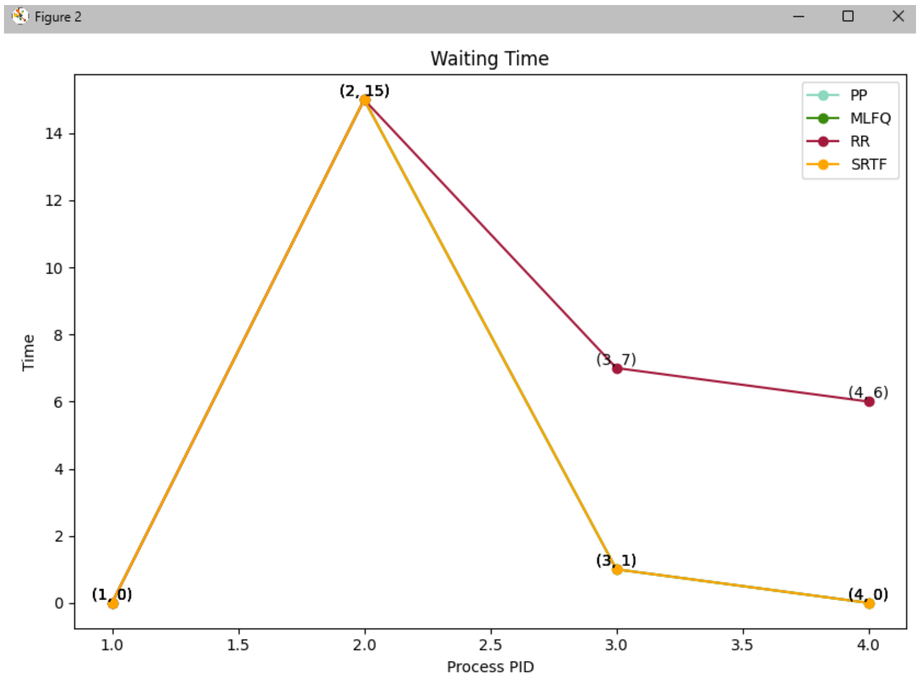
The Average Response Times

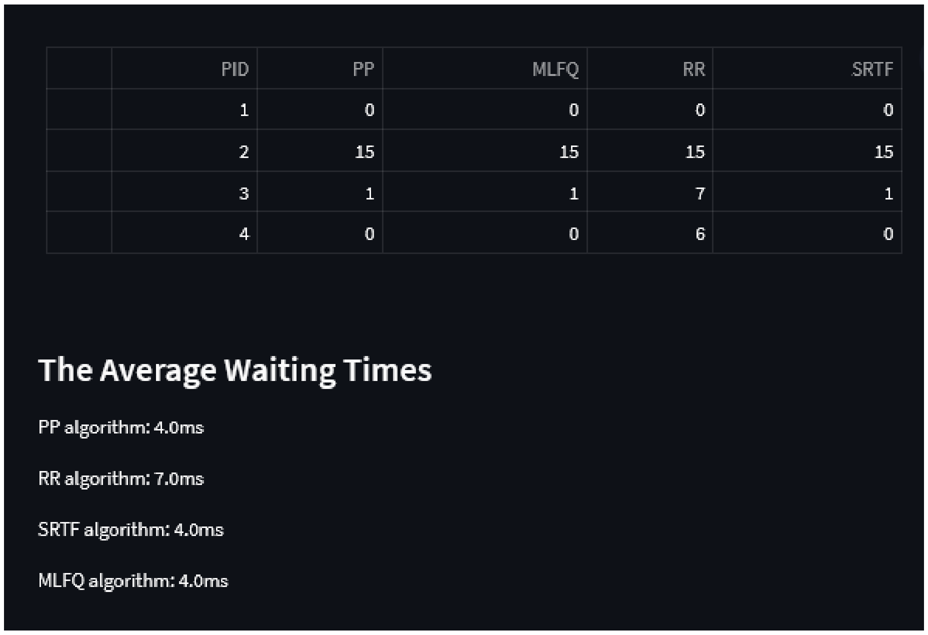
PP algorithm: 4.0ms

RR algorithm: 4.0ms

SRTF algorithm: 4.0ms

MLFQ algorithm: 4.0ms





	PID	PP	MLFQ	RR	SRTF
	1	0	0	0	0
	2	15	15	15	15
	3	1	1	7	1
	4	0	0	6	0

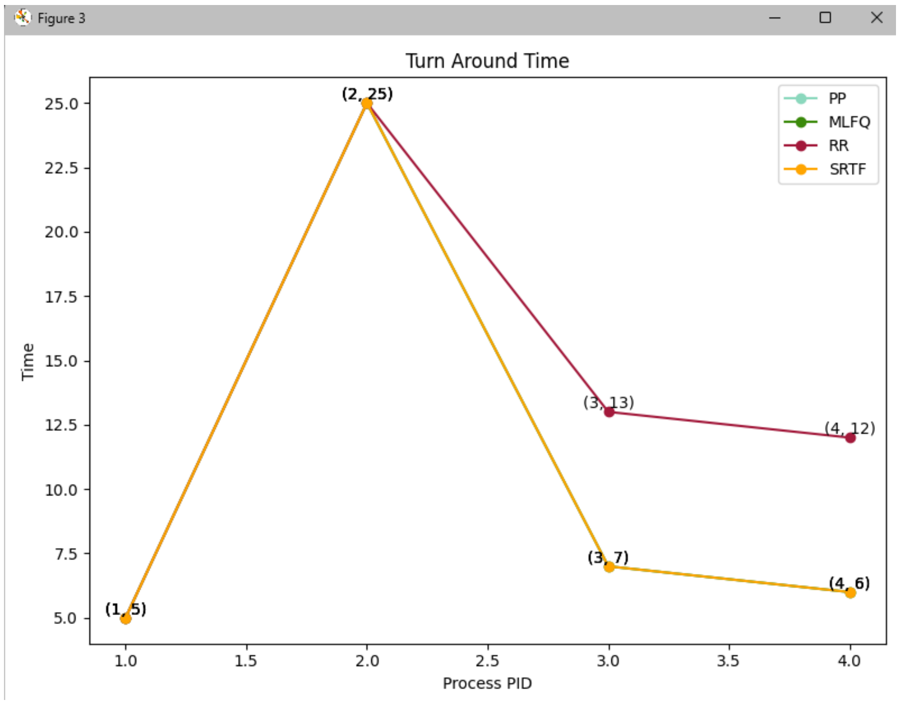
The Average Waiting Times

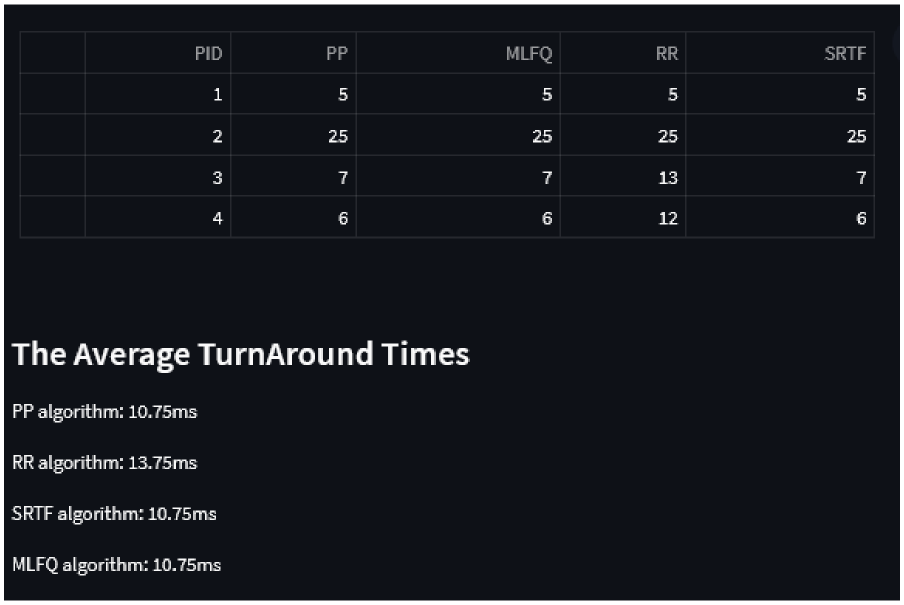
PP algorithm: 4.0ms

RR algorithm: 7.0ms

SRTF algorithm: 4.0ms

MLFQ algorithm: 4.0ms





	PID	PP	MLFQ	RR	SRTF
	1	5	5	5	5
	2	25	25	25	25
	3	7	7	13	7
	4	6	6	12	6

The Average TurnAround Times

PP algorithm: 10.75ms

RR algorithm: 13.75ms

SRTF algorithm: 10.75ms

MLFQ algorithm: 10.75ms

The performance results from the second input file demonstrate a smaller variance among the algorithms compared to those from the first input file. Notably, in this scenario, MLFQ, STRF, and PP show similar outcomes, leading to overlapping plots. This plot demonstrates that in some cases the algorithm selected does not affect the scheduling times for the process, that is why it is more optimal in these cases to select the algorithm with the lower cost in terms of complexity and implementation. Also, the algorithm RR is noticeably higher than the rest in terms of TAT and WT averages, this indicates that this algorithm is preforming poorly in this scenario.

Conclusion

Scheduling algorithms play a crucial role in CPU management, determining which processes are executed. Understanding the distinctions among the primary four algorithms (RR, PP, MLFQ, SRTF) helps developers in selecting the most suitable algorithm for their system. The tests conducted in this study revealed varying levels of performance differences among these algorithms. In certain scenarios, the differences were marginal, while in others, one algorithm, notably SRTF, show cases significant dominance.

SRTF emerges as the top performer, boasting the best Turnaround Time (TAT) and Waiting Time (WT). Following closely behind is MLFQ which consists of 3 main queues each implementing PP, RR, and FCFS algorithms, respectively. A tradeoff is inevitable in every system, in case the MLFQ was selected the system might cost more since the MLFQ depends on multiple queues, but the system will be able to run high priority processes without any context switches which might be critical for some systems.

MLFQ and PP offer enhanced system stability and control, whereas SRTF prioritizes minimal time consumption. On the other hand, RR ensures a fair CPU scheduling mechanism, devoid of any prioritization biases. Each algorithm presents a tradeoff, necessitating careful consideration based on specific system requirements and priorities.

References & Sources

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2012). Operating System Concepts. Ninth Edition. John Wiley & Sons, Inc.
2. Full source code can be found here.
3. The Streamlit API website can be found here.