VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI - 590018

Activity Report

Shortest Remaining Time First Scheduling

Submitted in partial fulfillment for the award of activity marks

Course Name : Operating Systems
Course code : 18CS43

Submitted by :

Prajwal R
1BG19CS071

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage,
Bengaluru- 560070, INDIA Ph: 91-80- 26711780/81/82 Email:
principal@bnmit.in, www. bnmit.org

Department of Computer Science and Engineering

2020 – 2021

Abstract

This report contains a brief description about Shortest Remaining Time First (SRTF) scheduling technique applied in CPU process scheduling. SRTF scheduling uses preemptiveness to decide which process has to be scheduled for execution. This concept is being explained in detail here with a suitable Java program demonstration.

ACKNOWLEDGEMENT

I am thankful to Jalaja G. Ma'am, Associate Professor, Department of Computer Science & Engg., for giving me an opportunity to report findings about the topic "Shortest Remaining Time First (SRTF) scheduling".

Prajwal R.

TABLE OF CONTENT

LIST OF TABLES

Chapter 1

Introduction

CPU Scheduling is a technique of determining which process shall own the CPU for execution while other processes are on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least selects one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

There are two types of Scheduling :

1) Preemptive Scheduling

2) Non - Preemptive Scheduling

Preemptive scheduling is a type of scheduling where the process is executed based on priority. This type of scheduling may pause the execution of the current process to start executing a more prioritized process thereby achieving time efficiency for shorter processes.

Non - Preemptive scheduling is a type of scheduling where the process in execution cannot be paused based on any kind of priority. A new process can only be executed after the completion of the previous process.

Some Terminologies to know :

CPU Burst time : It’s the number of CPU cycles required by the process to complete execution.
Arrival Time : It’s the CPU cycle at which the process arrives at the ready queue.
Turnaround time : It’s the Total number of cycles taken by the process to finish execution.

Shortest Remaining Time First scheduling is a Preemptive scheduling technique where the priority of the task is decided based on CPU Burst time remaining. The shorter the remaining Burst time, the higher priority it gets. It’s also called a Preemptive version of Shortest Job First Scheduling.

Advantage of Shortest Remaining Time First Scheduling is that the Time taken for execution of shorter period jobs is lower thereby increasing the responsiveness of shorter jobs compared to jobs that need longer time of execution. This is critical in systems that need to be highly responsive.

Disadvantage of this Scheduling technique is that a process with a longer time of execution may suffer Process Starvation : long processes may be held off indefinitely if short processes are continually added.

Chapter 2

SRTF Scheduling

Let's consider a problem to understand the Scheduling technique :

We need to schedule processes P1, P2, P3, P4 and P5. Their arrival time and burst time are given below in the table.

Process ID	Arrival Time	Burst Cycles
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

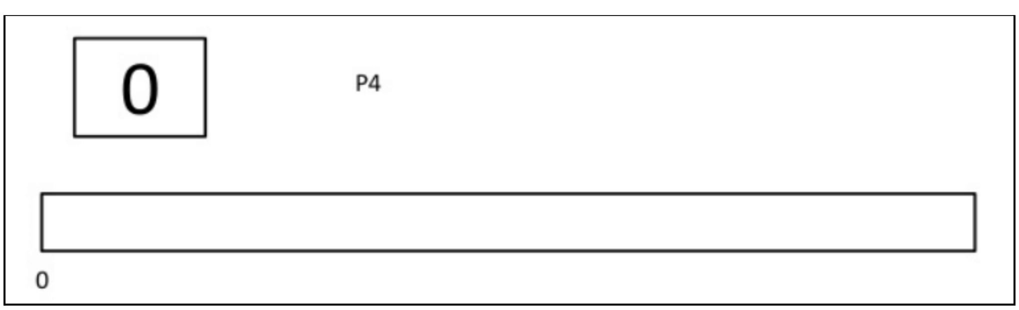
Table T(1)

The solution using SRTF scheduling is as follows :

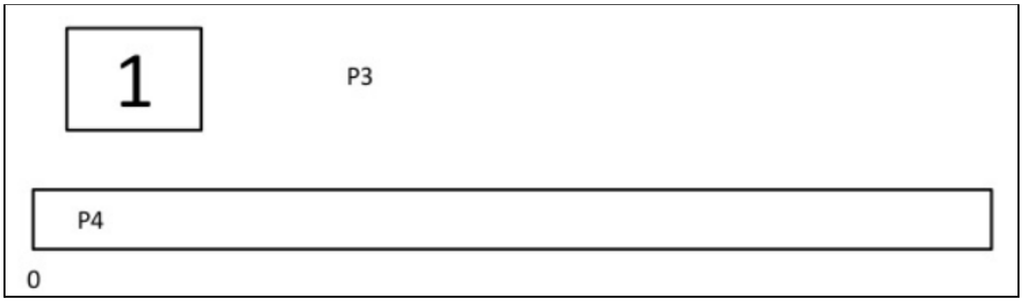
Step 0) At time=0, P4 arrives and starts execution.

Process ID	Arrival Time	Burst Cycles
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

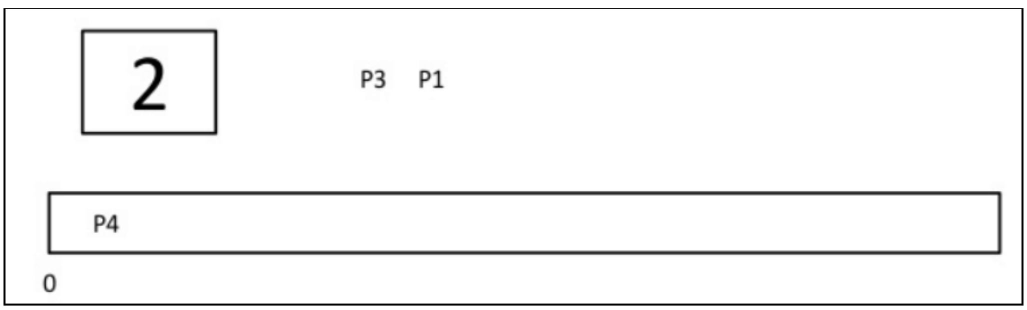
Table T(2)



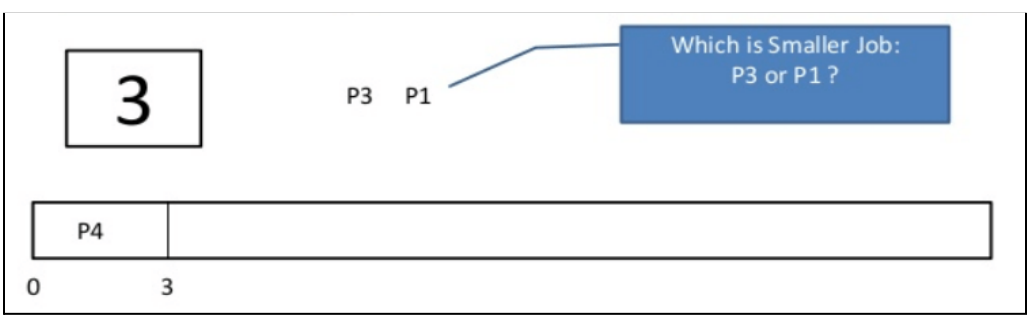
Step 1) At time= 1, Process P3 arrives. But, P4 has a shorter burst time. It will continue execution.



Step 2) At time = 2, process P1 arrives with burst time = 6. The burst time is more than that of P4. Hence, P4 will continue execution.



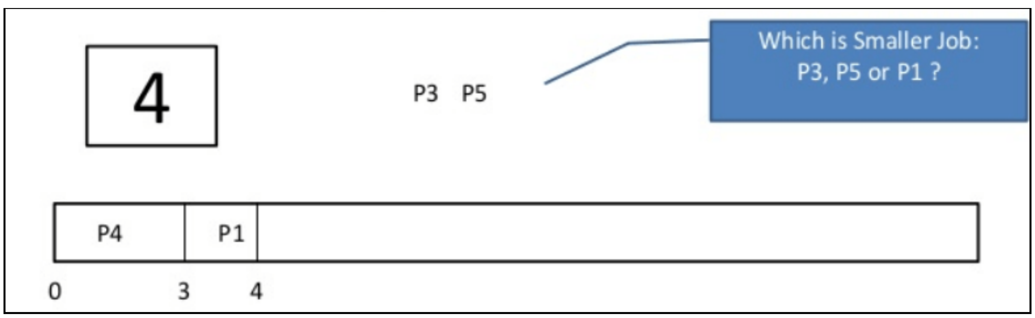
Step 3) At time = 3, process P4 will finish its execution. The burst time of P3 and P1 is compared. Process P1 is executed because its burst time is lower.



Step 4) At time = 4, process P5 will arrive. The burst time of P3, P5, and P1 is compared. Process P5 is executed because its burst time is lowest. Process P1 is preempted.

Process ID	Arrival Time	Burst Cycles
P1	2	5 out of 6 is remaining
P2	5	2
P3	1	8
P4	0	3
P5	4	4

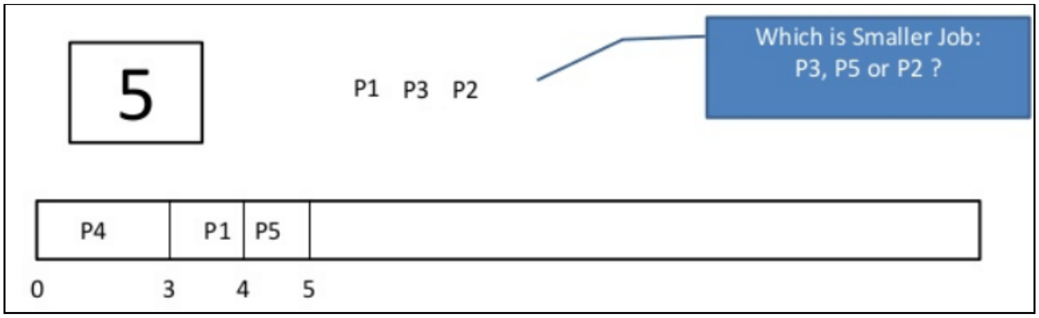
Table T(3)



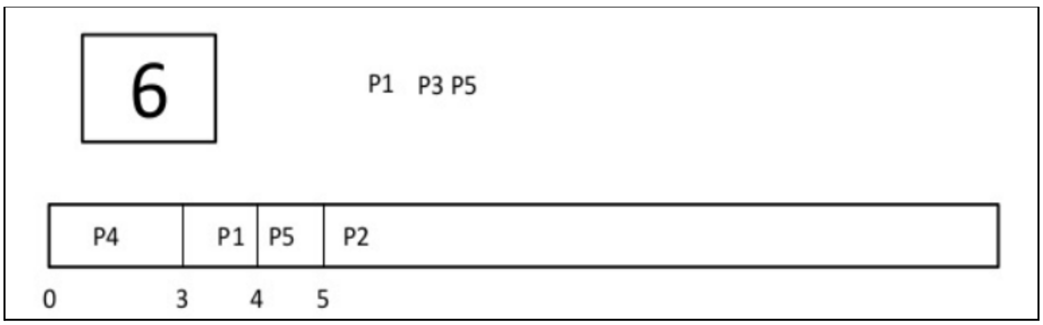
Step 5) At time = 5, process P2 will arrive. The burst time of P1, P2, P3, and P5 is compared. Process P2 is executed because its burst time is least. Process P5 is preempted.

Process ID	Arrival Time	Burst Cycles
P1	2	5 out of 6 is remaining
P2	5	2
P3	1	8
P4	0	3
P5	4	3 out of 4 is remaining

Table T(4)



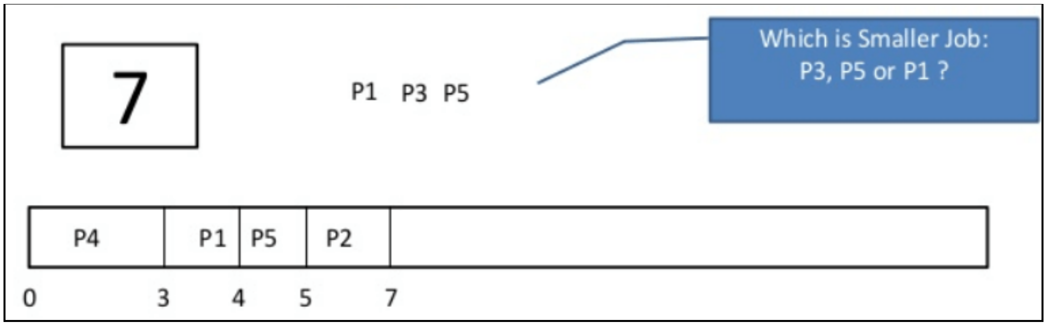
Step 6) At time =6, P2 is executing.



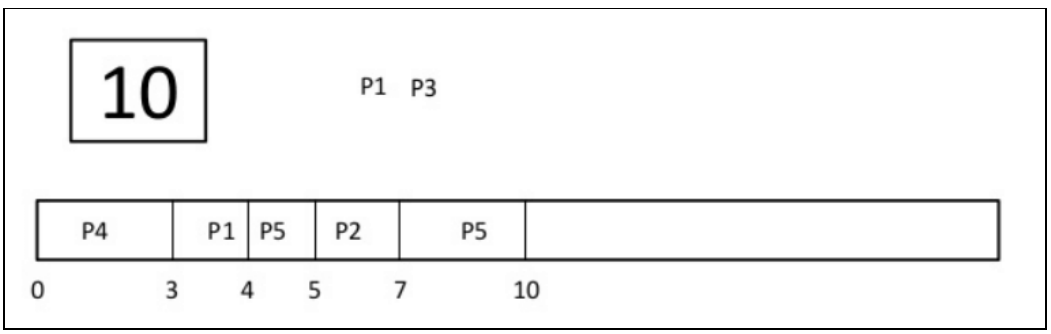
Step 7) At time =7, P2 finishes its execution. The burst time of P1, P3, and P5 is compared. Process P5 is executed because its burst time is lesser.

Process ID	Arrival Time	Burst Cycles
P1	2	5 out of 6 is remaining
P2	5	2
P3	1	8
P4	0	3
P5	4	3 out of 4 is remaining

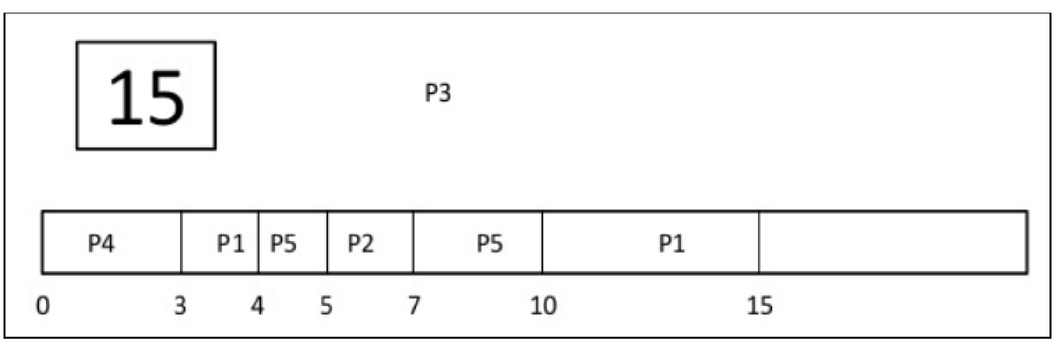
Table T(5)



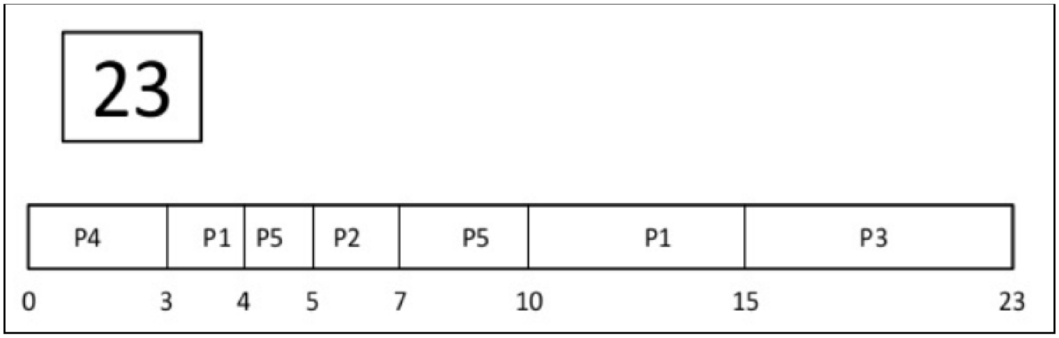
Step 8) At time =10, P5 will finish its execution. The burst time of P1 and P3 is compared. Process P1 is executed because its burst time is less.



Step 9) At time =15, P1 finishes its execution. P3 is the only process left. It will start execution.



Step 10) At time =23, P3 finishes its execution.



Step 11) Let's calculate the average waiting time for the above example.

- Turn Around time = Exit time – Arrival time
- Waiting time = Turnaround time – Burst time

Turn Around time
P4 = 3 - 0 = 3
P1 = 15 - 2 = 13
P2 = 7 - 5 = 2
P5 = 10 - 4 = 6
P3 = 23 - 1 = 22

Average Turnaround time = (3+13+2+6+22)/5 = 46/5 = 9.2

Wait time
P4 = 0 - 0 = 0
P1 = (3 - 2) + 6 = 7
P2 = 5 - 5 = 0
P5 = 4 - 4 + 2 = 2
P3 = 15 - 1 = 14

Average Waiting Time = 0+7+0+2+14/5 = 23/5 = 4.6

Chapter 3

SRTF Java Program & Outputs

Program :

```
/* Shortest Remaining Time First Job Scheduling Program for OS Activity */
import java.util.Scanner;

class Job
{
    String jobName;
    int cpuCycles, arrivalTime;
    int burstTime;
    int endPoint = 0;
    int flag = 0;
    Job(String jobName, int cpuCycles, int arrivalTime)
    {
        this.jobName = jobName;
        this.cpuCycles = cpuCycles;
        this.arrivalTime = arrivalTime;
        this.burstTime = cpuCycles;
    }
    public void execution(int cpuCycleTime)
    {
        System.out.println("CPU Cycle " + cpuCycleTime + " : Process " + jobName + "
in execution");
        if(cpuCycles > 1)
            cpuCycles--;
        else
            flag = 1;
    }
}

public class SRTFProgram
{
    static int top = -1;
    public static void main(String args[])
    {
        Scanner read = new Scanner(System.in);
        int numberOfJobs;
```

```
        System.out.println("Enter the number of Jobs to be Scheduled ");
        numberOfJobs = read.nextInt();

        System.out.println("Enter the Name and CPU Cycles for the jobs : ");
        Job process[] = new Job[numberOfJobs];

        int totalTimeOfExe = 0;

        for(int param = 0; param < numberOfJobs; param++)
        {
            System.out.print("Enter name : ");
            String jobName = read.next();

            System.out.print("Enter CPU Burst Cycles : ");
            int cpuCycles = read.nextInt();

            System.out.print("Enter Arrival Time : ");
            int arrivalTime = read.nextInt();

            process[param] = new Job(jobName,cpuCycles,arrivalTime);
            totalTimeOfExe += cpuCycles;
        }

        System.out.println("\nThe jobs are :");
        for(int param1 = 0; param1 <numberOfJobs; param1++)
            System.out.println("Job Name : " + process[param1].jobName + "\t CPU Burst
Cycles : " + process[param1].cpuCycles + "\t Arrival Time : " +
process[param1].arrivalTime);

        System.out.println("\nTotal Time of execution : " +totalTimeOfExe);

        int cpuCycleCount = 0;
        Job jobStack[] = new Job[numberOfJobs];
        int count = 0;

        Job processDone[] = new Job[numberOfJobs];

        while(cpuCycleCount < totalTimeOfExe)
        {
```

```
            for(int param = 0; param< numberOfJobs; param++)
                {
                    if(process[param].arrivalTime == cpuCycleCount)
                        {
                            push(jobStack, process[param]);
                        }
                }

            for(int param = 0; param <= top; param++)
               {
                 for(int param2 = 0; param2<= top-param-1; param2++)
                   {

                        if(jobStack[param2].cpuCycles <= jobStack[param2+1].cpuCycles)
                        {
                            Job temp_var;
                            temp_var = jobStack[param2];
                            jobStack[param2] = jobStack[param2+1];
                            jobStack[param2+1] = temp_var;
                        }
                   }
               }

            cpuCycleCount++;

            jobStack[top].execution(cpuCycleCount);

            if(jobStack[top].flag == 1)
               {
                  jobStack[top].endPoint = cpuCycleCount;
                  processDone[count++] = pop(jobStack);
               }
        }

        System.out.println("\nThe Pattern of Scheduling is :");
        int turnAroundTime = 0;
        float waitingTime = 0;

        for(int param1 = 0; param1 < numberOfJobs; param1++)
        {
```

```
            int turnAround = processDone[param1].endPoint -
processDone[param1].arrivalTime;
            int waiting = turnAround-processDone[param1].burstTime;
            System.out.println("Job Name : " + processDone[param1].jobName + "\t Turn
Around time : " + turnAround + "\t Waiting Time : " + waiting);
            turnAroundTime += turnAround;
            waitingTime += waiting;
        }

        System.out.println("Avg Turn Around time : " + (turnAroundTime/5));
        System.out.println("Avg Waiting time : " + (waitingTime/5));
    }

    static void push(Job jobStack[],Job  item)
    {
            jobStack[++top] = item;
    }

    static Job pop(Job jobStack[])
    {
        Job temp_obj;
        if(top == 0)
        {
            temp_obj = jobStack[top];
            top = -1;
        }
        else
        {
            temp_obj = jobStack[top--];
        }
        return temp_obj;
    }
}
```

Output :

Using the same inputs as the given problem.

```
Enter the number of Jobs to be Scheduled
5
Enter the Name and CPU Cycles for the jobs :
Enter name : P1
Enter CPU Burst Cycles : 6
Enter Arrival Time : 2
Enter name : P2
Enter CPU Burst Cycles : 2
Enter Arrival Time : 5
Enter name : P3
Enter CPU Burst Cycles : 8
Enter Arrival Time : 1
Enter name : P4
Enter CPU Burst Cycles : 3
Enter Arrival Time : 0
Enter name : P5
Enter CPU Burst Cycles : 4
Enter Arrival Time : 4

The jobs are :
Job Name : P1    CPU Burst Cycles : 6    Arrival Time : 2
Job Name : P2    CPU Burst Cycles : 2    Arrival Time : 5
Job Name : P3    CPU Burst Cycles : 8    Arrival Time : 1
Job Name : P4    CPU Burst Cycles : 3    Arrival Time : 0
Job Name : P5    CPU Burst Cycles : 4    Arrival Time : 4

Total Time of execution : 23
CPU Cycle 1 : Process P4 in execution
CPU Cycle 2 : Process P4 in execution
CPU Cycle 3 : Process P4 in execution
CPU Cycle 4 : Process P1 in execution
CPU Cycle 5 : Process P5 in execution
CPU Cycle 6 : Process P2 in execution
CPU Cycle 7 : Process P2 in execution
CPU Cycle 8 : Process P5 in execution
CPU Cycle 9 : Process P5 in execution
CPU Cycle 10 : Process P5 in execution
CPU Cycle 11 : Process P1 in execution
CPU Cycle 12 : Process P1 in execution
CPU Cycle 13 : Process P1 in execution
CPU Cycle 14 : Process P1 in execution
CPU Cycle 15 : Process P1 in execution
```

```
CPU Cycle 16 : Process P3 in execution
CPU Cycle 17 : Process P3 in execution
CPU Cycle 18 : Process P3 in execution
CPU Cycle 19 : Process P3 in execution
CPU Cycle 20 : Process P3 in execution
CPU Cycle 21 : Process P3 in execution
CPU Cycle 22 : Process P3 in execution
CPU Cycle 23 : Process P3 in execution

The Pattern of Scheduling is :
Job Name : P4    Turn Around time : 3    Waiting Time : 0
Job Name : P2    Turn Around time : 2    Waiting Time : 0
Job Name : P5    Turn Around time : 6    Waiting Time : 2
Job Name : P1    Turn Around time : 13   Waiting Time : 7
Job Name : P3    Turn Around time : 22   Waiting Time : 14
Avg Turn Around time : 9.2
Avg Waiting time : 4.6
```

Conclusion

Shortest Remaining Time First Scheduling is a CPU process scheduling technique that helps us design an operating system for embedded systems. It's higher responsiveness to shorter jobs makes it very suitable for embedded system applications like controllers for Washing Machines, Refrigerators, etc.

References

1) OS SRTF scheduling Algorithm - javatpoint
2) Shortest Job First (SJF): Preemptive, Non-Preemptive Example (guru99.com)

****