Analysis of CPU Scheduling Algorithms

Operating Systems
CSD-222

Submitted by: Kashish Srivastava (185014)
Dipesh Kumar (185015)
Akash Rana (185034)

CSE (4 Year) : 4th Semester

Under the guidance of

Dr. Pradeep Singh
Assistant Professor
Department of Computer Science and Engineering
National Institute of Technology, Hamirpur

Introduction

This document is a report for the individual project “Simulation of various CPU scheduling algorithms and analysing the behaviour of each scheduler”. CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

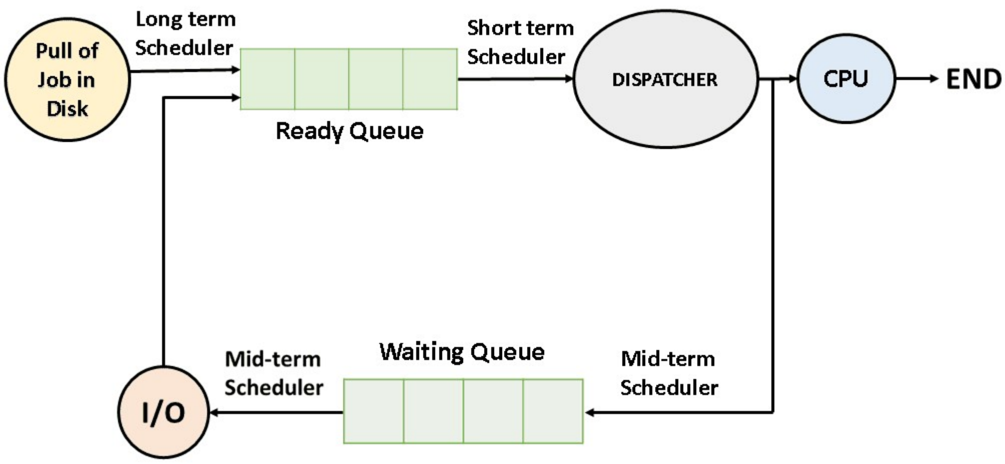


Figure 1.1: Cpu scheduling

The project is an attempt to differentiate the behaviour of each scheduler with the statistical approach. In this project we performed each CPU scheduling and compared them on the basis of their completion time,throughput,average job elapsed time and average job waiting time during the process. It also discusses the complexity of the written code as instructed.

Software requirement and specification

Python(3.7.6)

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

Python-dateutil(2.8.1)

The dateutil module provides powerful extensions to the standard datetime module, available in Python.

matplotlib(3.2.2)

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

cycler(0.10.0)

A single entry Cycler object can be used to easily cycle over a single style.

numpy(1.19.0)

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices).

Kiwisolver(1.2.0)

Kiwisolver is an efficient C++ implementation of the Cassowary constraint solving algorithm. Kiwi is an implementation of the algorithm based on the seminal Cassowary paper.

pyparsing(2.4.7)

Pyparsing is a mature, powerful alternative to regular expressions for parsing text into tokens and retrieving or replacing those tokens.

six(1.15.0)

Six provides simple utilities for wrapping over differences between Python 2 and Python 3. It is intended to support codebases that work on both Python 2 and 3 without modification.

Usage

Install Python 3.7.6 and then install all packages mentioned in 'requirements.txt' in root folder by:

```
python -m pip install -r requirements.txt
```

After Installing all the packages run the main script with command:

```
python main.py
```

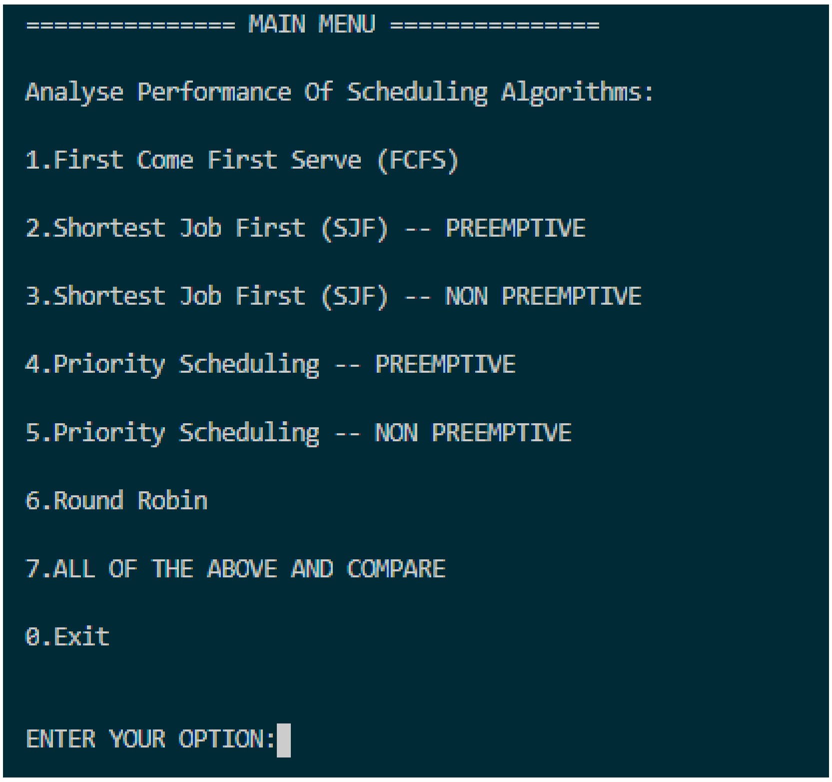


Figure 3.1: Main Menu

Contents

First Come First Serve (FCFS)

Listing 4.1: fcfs source code

```
import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')

## Finds waiting time for each process
def findWaitingTime(processes,  burst_time, waiting_time, arrival_time):
    service_time = [0] *len(processes)
    service_time[0] = 0
    waiting_time[0] = 0
    for i in range(1, len(processes)):

        service_time[i] = (service_time[i - 1] +  burst_time[i - 1])

        waiting_time[i] = service_time[i] - arrival_time[i]
        if (waiting_time[i] < 0):
            waiting_time[i] = 0


#Finds Turn Around Time for All processes
def findTurnAroundTime(processes, burst_time, waiting_time, turn_around_time)
    for i in range(len(processes)):
        turn_around_time[i] = burst_time[i] + waiting_time[i]


#Returns waiting,turn around, completion time for each process
def findavgTime(processes,  burst_time, arrival_time):

    waiting_time = [0] *len(processes)
    turn_around_time = [0] *len(processes)
    findWaitingTime(processes,  burst_time, waiting_time, arrival_time)
    findTurnAroundTime(processes,  burst_time, waiting_time, turn_around_time

    total_wt = 0
    total_turn_around_time = 0
    compl_time = [0]*len(processes)
    for i in range(len(processes)):

        total_wt = total_wt + waiting_time[i]
        total_turn_around_time = total_turn_around_time + turn_around_time[i]
        # Calculate completion time
```

```
        compl_time[i] = turn_around_time[i] + arrival_time[i]

    return waiting_time , turn_around_time ,compl_time


def plot_graph(processes,waiting_time,compl_time,turn_around_time):

    plt.plot(processes,waiting_time,label = "Waiting time")
    plt.plot(processes,compl_time,label = "Completion time")
    plt.plot(processes,turn_around_time,label = "Turnaround Time")
    plt.text(4,2,'Throughput = %.5f' % (len(processes)/ compl_time[len(proce
    plt.title("First Come First Serve Algo")
    plt.xlabel("Processes")
    plt.ylabel("Time Units")
    plt.legend()
    plt.savefig('./output/FCFS_output.png')
    plt.show()



def print_details(processes,waiting_time,turn_around_time,compl_time,burst_tin

    print("Processes   Burst Time   Arrival Time     Waiting",
          "Time   Turn-Around Time  Completion Time \n")
    for i in range(len(processes)):
        print(" ", processes[i] , "\t\t", burst_time[i], "\t\t", arrival_time
               "\t\t", waiting_time[i], "\t\t ", turn_around_time[i], "\t\t ",

    print("Average waiting time = %.5f "%(sum(waiting_time) /len(processes)))
    print("\nAverage turn around time = ", sum(turn_around_time) / len(proces
    print('\nThroughput = ', len(processes)/ compl_time[len(processes)-1])


# driver function

def fcfs():

    processes = []
    burst_time = []
    arrival_time = []
    #breakpoint
    with open('./inputs/FCFS.txt','r') as f:
        f.readline()
```

```
        for line in f.readlines():
            process , burst , arrival = (line.split(" "))
            processes.append(process)
            burst_time.append(int(burst))
            arrival_time.append(int(arrival))

    #returned waiting time and turn around time
    waiting_time , turn_around_time, compl_time = findavgTime(processes,
burst_time, arrival_time)

    #print details about data
    print_details(processes,waiting_time,turn_around_time,compl_time,burst_tin

    #plotting
    plot_graph(processes,waiting_time,compl_time,turn_around_time)
    plt.close(fig='all')

if __name__ =="__main__":
    fcfs()
```

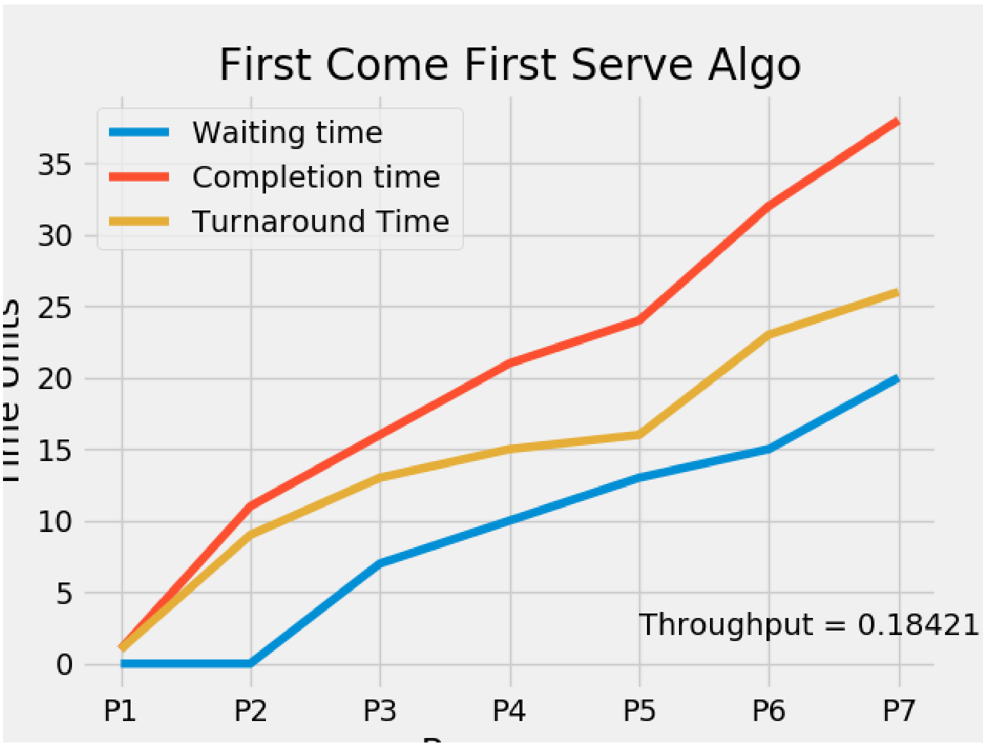


Figure 4.1: FCFS output values.

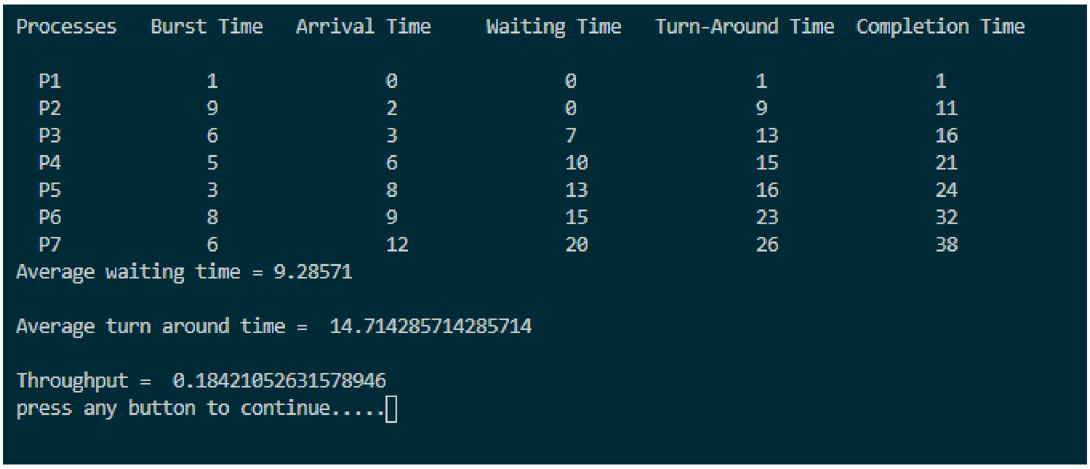


Figure 4.2: FCFS output values.

Shortest Job First(SJF)

5.1 SJF(Preemptive)

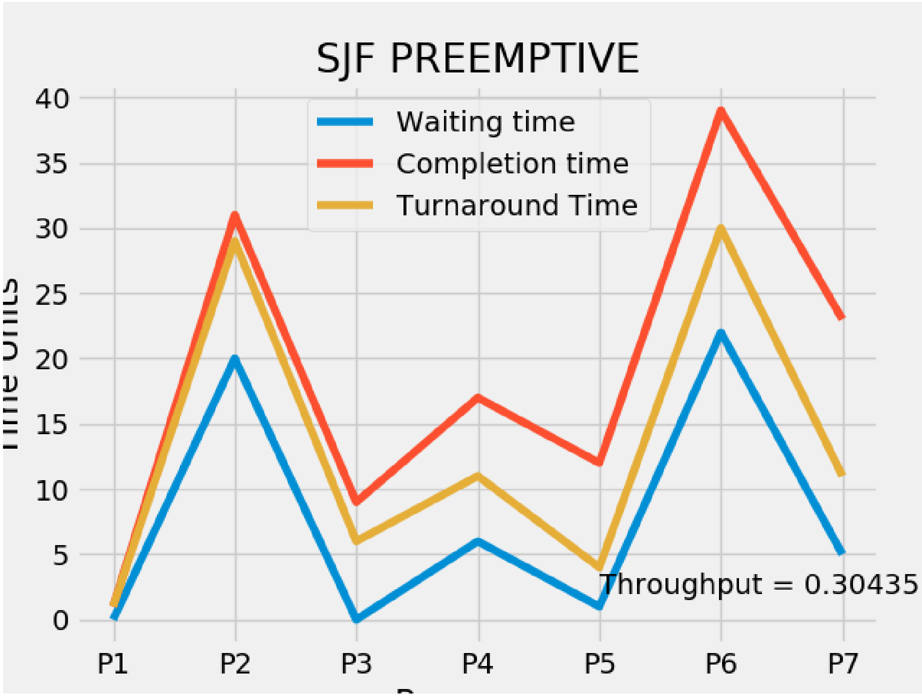
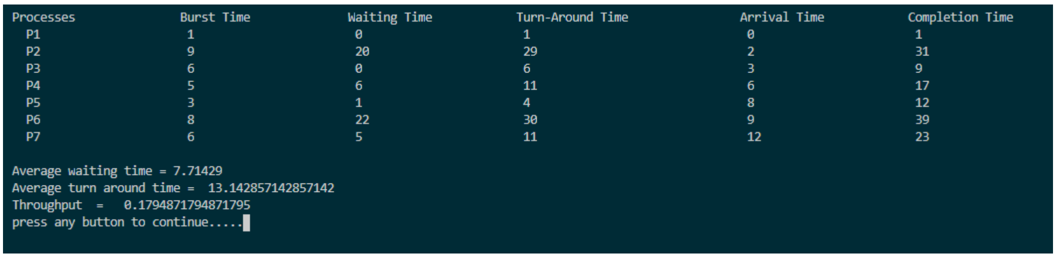


Figure 5.1: SJF(Preemptive) output values.



Processes	Burst Time	Waiting Time	Turn-Around Time	Arrival Time	Completion Time
P1	1	0	1	0	1
P2	9	20	29	2	31
P3	6	0	6	3	9
P4	5	6	11	6	17
P5	3	1	4	8	12
P6	8	22	30	9	39
P7	6	5	11	12	23

Average waiting time = 7.71429
Average turn around time = 13.142857142857142
Throughput = 0.1794871794871795
press any button to continue.....

Figure 5.2: SJF(Preemptive) output values.

5.2 SJF(Non-Preemptive)

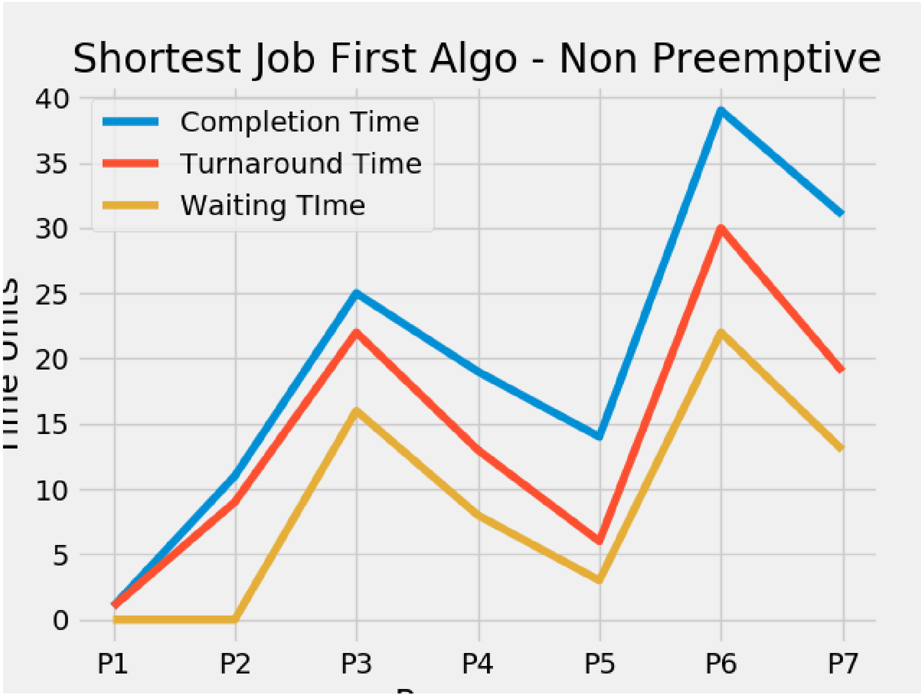
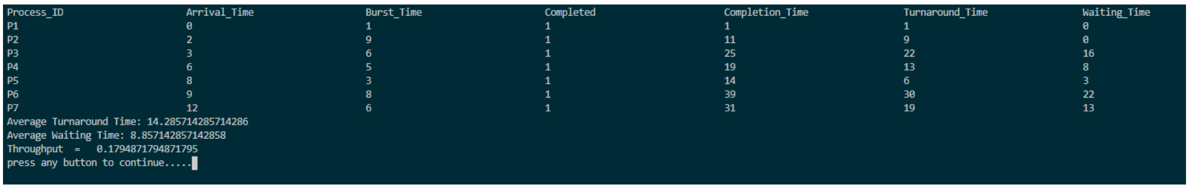


Figure 5.3: SJF(Preemptive) output values.



Process_ID	Arrival_Time	Burst_Time	Completed	Completion_Time	Turnaround_Time	Waiting_Time
P1	0	1	1	1	1	0
P2	2	9	1	11	9	0
P3	3	6	1	25	22	16
P4	6	5	1	19	13	8
P5	8	3	1	14	6	3
P6	9	8	1	39	30	22
P7	12	6	1	31	19	13

Average Turnaround Time: 14.285714285714286
Average Waiting Time: 8.857142857142858
Throughput = 0.1794871794871795
press any button to continue.....

Figure 5.4: SJF(Preemptive) output values.

Priorty Scheduling

6.1 Priorty(Preemptive)

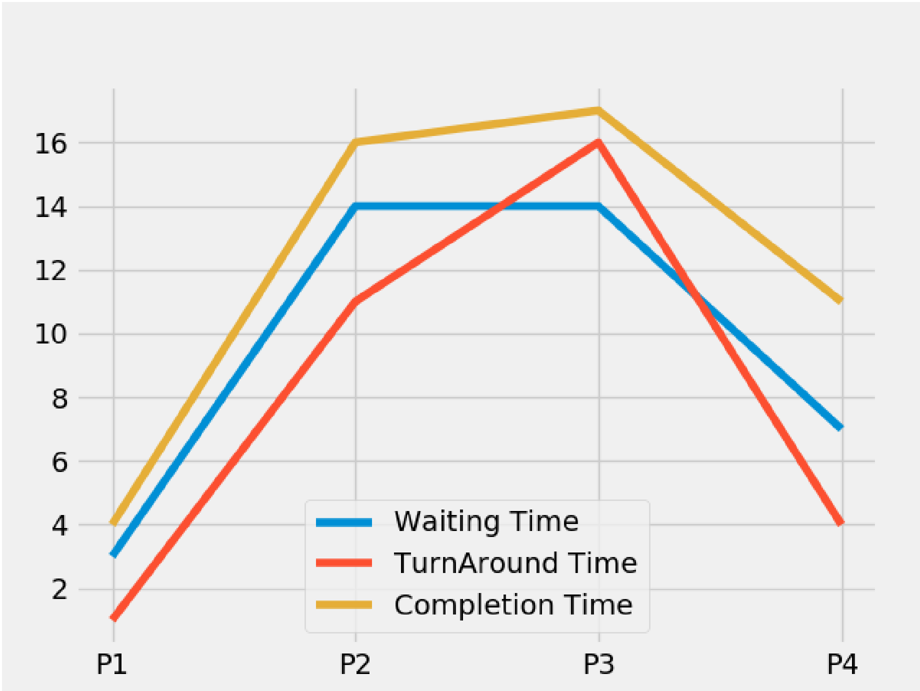


Figure 6.1: SJF(Preemptive) output values.

```
Process_ID  Arrival_Time  Rem_Burst_Time   Priority       Completed  Orig_Burst_Time Completion_Time  Turnaround_Time  Waiting_Time
P1               1             0              8               1            3              4                1               3
P2               2             0              5               1            5              16               11              14
P3               3             0              1               1            1              17               16              14
P4               4             0              7               1            7              11               4               7

Average Turnaround Time: 8.0
Average Waiting Time: 9.5
Throughput:  0.23529411764705882
Sequence of Process: ([3, 14, 14, 7], [1, 11, 16, 4], [4, 16, 17, 11], ['P1', 'P1', 'P1', 'P4', 'P4', 'P4', 'P4', 'P4', 'P4', 'P4',
press any button to continue.....
```

Figure 6.2: SJF(Preemptive) output values.

6.2 Priorty(Non-Preemptive)

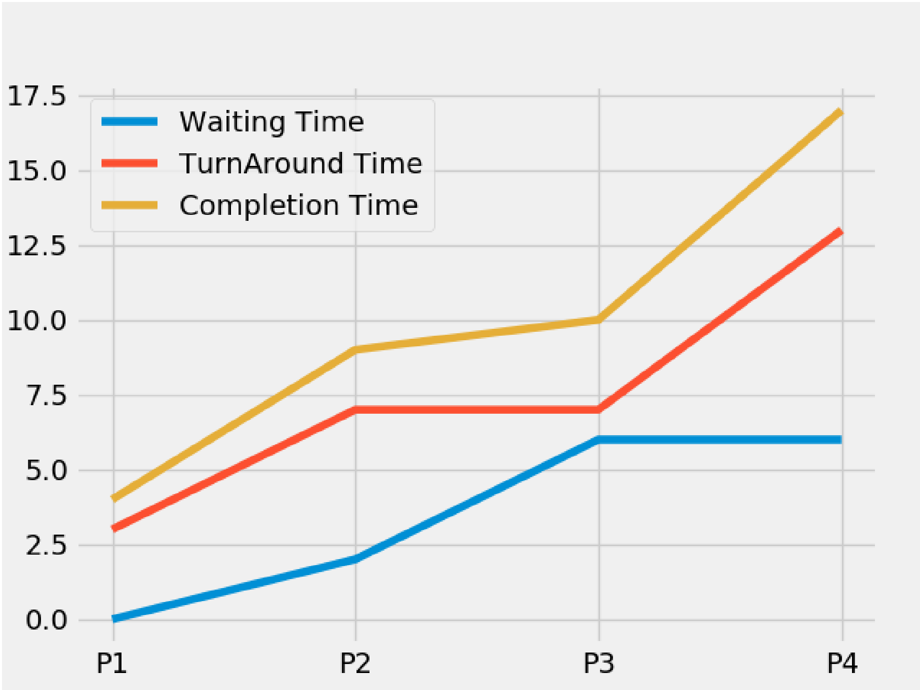


Figure 6.3: SJF(Preemptive) output values.

Process_no	Start_time	Complete_time	Turn_Around_Time	Waiting_Time	Priority
P1	1	4	3	0	2
P2	2	9	7	2	5
P3	3	10	7	6	1
P4	4	17	13	6	7

```
Average waiting time is :  3.5
average turnaround time :  7.5
press any button to continue.....
```

Figure 6.4: SJF(Preemptive) output values.

Round Robin

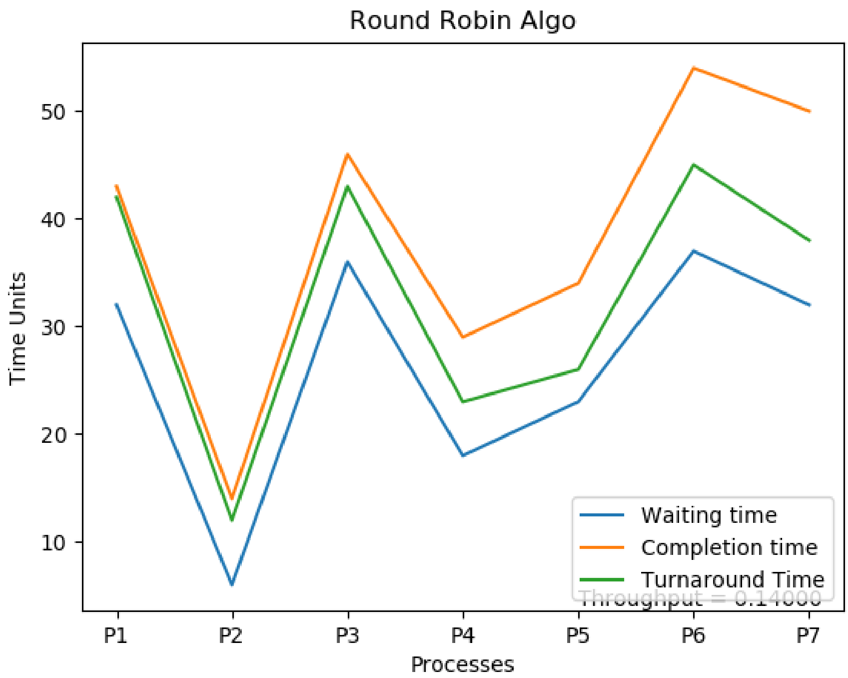


Figure 7.1: FCFS output values.

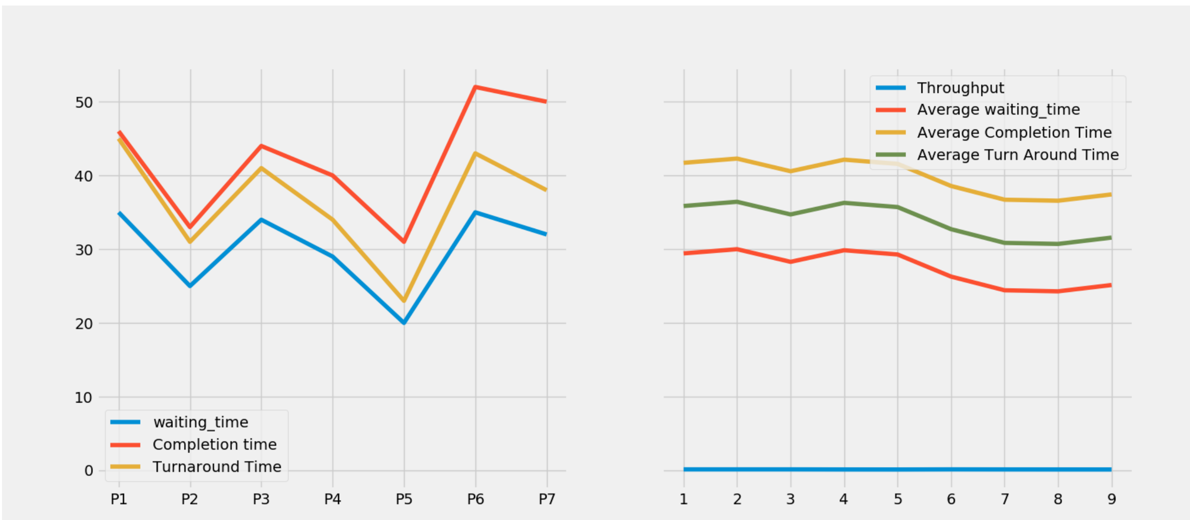


Figure 7.2: FCFS output values.

```
Processes     Burst Time      Waiting  Time     Turn-Around Time      Completion Time
  P1             10              35               45                  46
  P2             6               25               31                  33
  P3             7               34               41                  44
  P4             5               29               34                  40
  P5             3               20               23                  31
  P6             8               35               43                  52
  P7             6               32               38                  50

Average waiting_time = 30.00000
Average turn around time = 36.42857
Throughput =  0.1346153846153846
Average Job elapsed time =  36.42857142857143
press any button to continue.....
```

Figure 7.3: FCFS output values.

Conclusion

From the given project work we have concluded that the difference between the turn around time,average waiting time,average elapsed time and completion time for same input of different cpu scheduling is as follows:-

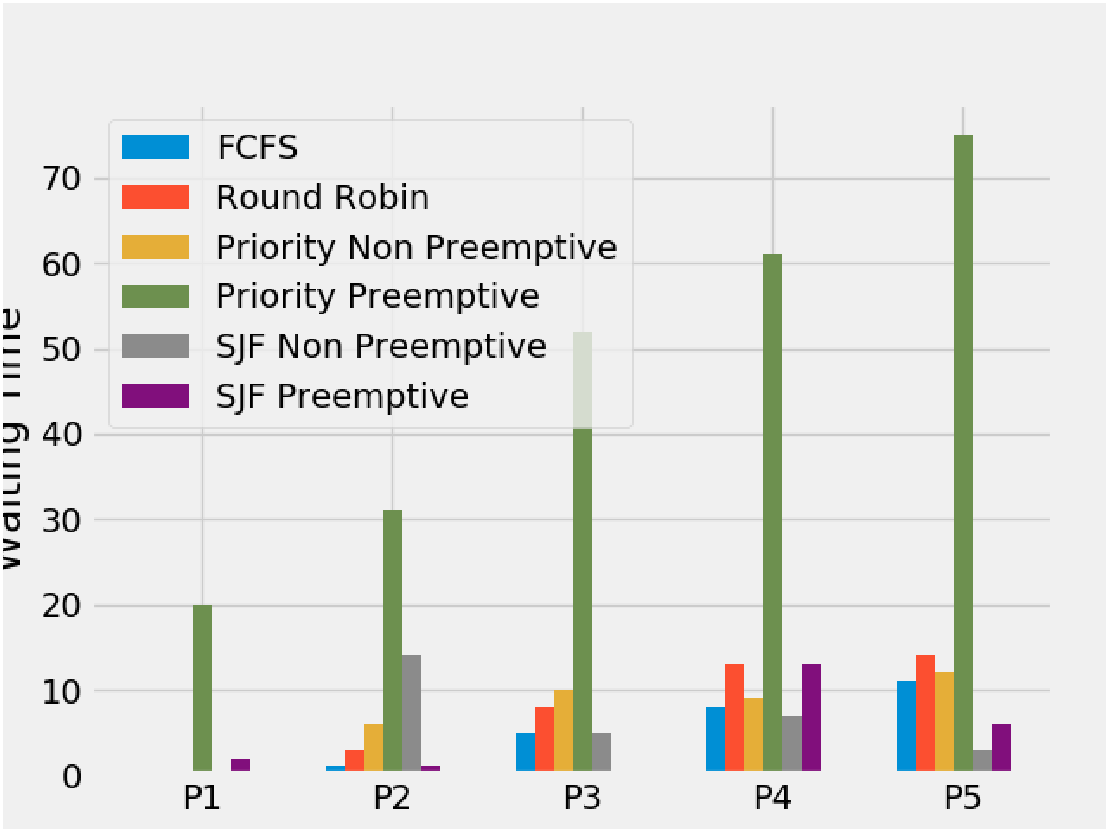


Figure 8.1: comparison between CPU scheduling.

from the above graph we came to know that every algorithm works better on the significant problem as the fcfs is better for a small burst time.The sjf is better if the process comes to processor simultaneously and round robin, is better to adjust the average waiting time desired and the priorty works better where the relative important of each process may be precisely defined.

References

The source code of the project can be found at:-
https://github.com/cannibalcheeseburger/cpu-scheduling-simluation.git
And the other references we used for this project are given:-

- A. Dusseau, R. H. dan A. C., Operating Systems: Three Easy Pieces, Arpaci-Dusseau Books, 2014.
- Operating System Principles – Galvin
- Tanenbaum, Modern Operating Systems, Pearson Education, Inc., 2008.
- http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html
- http://codex.cs.yale.edu/avi/os-book/OS8/os8c/slide-dir/PDF-dir/ch5.pdf