Visualization of CPU Scheduling Algorithms

(J-Component)

Project Report

Submitted by:-

Pranav V A(18BCE0748)
Vrinda Chopra(18BCE0785)

Course Code: 2005

Slot: F2

Course Title:
Operating System

In partial fulfilment for the award of the degree of

B.Tech in
Computer Science and Engineering

Under the guidance of

Dr. Sendhil Kumar K S, VIT, Vellore

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

Our work aims at providing a thorough analysis of various CPU Scheduling Algorithms namely, FCFS (First Come First Serve), SJF (Shortest Job First) and Round Robin.
The program takes input in the form of various data files such as an Excel file or Word file. The input contains process IDs, burst times and arrival times. The program then forms an appropriate ready queue starts executing the processes. The simulator first reads task information from input file and stores all data in a data structure. Then it starts simulating one scheduling algorithm in a time-driven manner. At each time unit (or slot), it adds any newly arrived task(s) into the ready queue and calls a specific scheduler algorithm in order to select appropriate task from ready queue. When a task is chosen to run, the simulator prints out a message indicating what process ID is chosen to execute for this time slot. If no task is running (i.e. empty ready queue), it prints out an "idle" message. Before advancing to the next time unit, the simulator should update all necessary changes in task and ready queue status.
The analysis is done on the following attributes

- waiting time
- throughput time

The crux of the project is the visualization of the processes which it achieves with the use of python libraries.
The visualization consists of a Gantt chart and waiting time graph.

INNOVATION

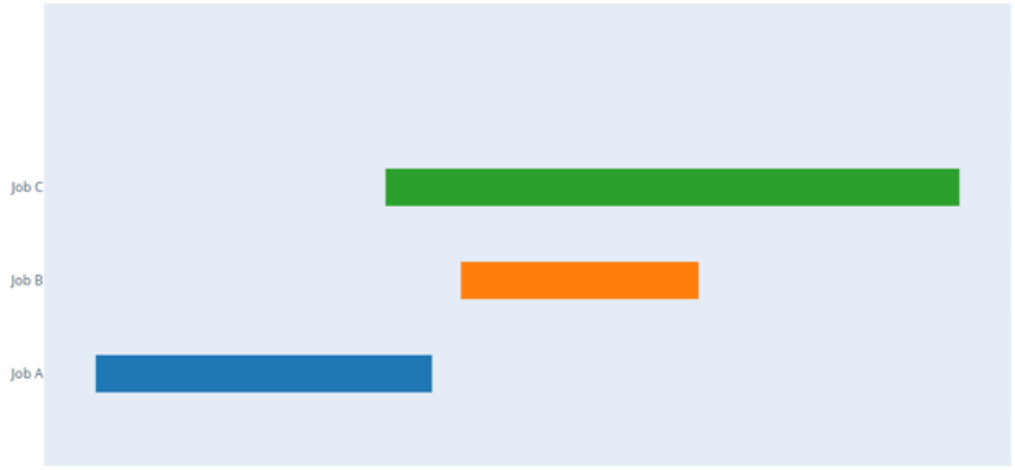
Out of so many algorithms its essential for us to analyse them in different environments, Modern computing is present almost everywhere and operating systems are an important part of it, we have OS present in almost all technological gadgets from our mobile phones (example android) to the touch screens of modern cars. Scheduling remains a common factor in all of these.

Various projects have been made on CPU scheduling algorithms but none have given the user a visual understanding of what is happening in the scene. Hence, we have used Gantt charts and graphs to solidify the understanding of CPU scheduling algorithms for the user. The project aims in not only providing data analysis of the scheduling algorithms but also graphical and pictorial representations, after thorough research we have found suitable tools to do the same in the form of python libraries.

It can be implemented in the following way:-

```
import plotly.figure_factory as ff

df = [dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28'),
      dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15'),
      dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30')]

fig = ff.create_gantt(df)
fig.show()
```

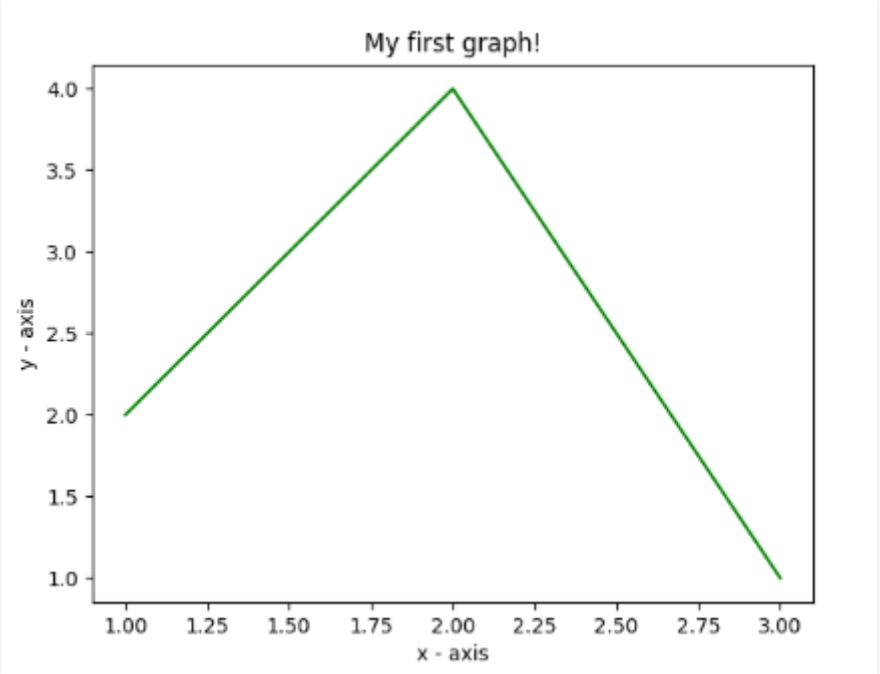
where plotly is the python library which allows us to create Gantt charts.




```
# importing the required module
import matplotlib.pyplot as plt
# x axis values
x = [1,2,3]
```

```
# corresponding y axis values
y = [2,4,1]
# plotting the points
plt.plot(x, y)
# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('My first graph!')
# function to show the plot
plt.show()
```

This is an example of the graph will be drawn giving a comparison of how the particular algorithm performs in a certain environment.



The above is an example of a plotted graph related to one function, here we can plot multiple functions at once this can be used for easy comparison between the performance's.

IMPLEMENTATION

a) FCFS

CODE

```
# Importing of libraries

import xlrd
import plotly.figure_factory as ff
import matplotlib.pyplot as plt
import numpy as np

process = "Process "

# Excel file linked

loc = ("Book1.xlsx")

wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)
sheet.cell_value(0, 0)

# Burst_time array initialized and finding of process list

Dict1 = {}
Dict2 = {}
for i in range(1,sheet.nrows):
    Dict1[sheet.cell_value(i,2)] = sheet.cell_value(i,1)
for i in range(1,sheet.nrows):
    Dict2[sheet.cell_value(i,2)] = sheet.cell_value(i,0)
processes1 = []
processes2 = []
processes1 = sorted(Dict1.keys())
processes2 = sorted(Dict2.keys())
proc = []
bt = []
for i in range(len(processes1)):
    bt.append(int(Dict1[processes1[i]]))
for i in range(len(processes2)):
    proc.append(int(Dict2[processes2[i]]))

# Creation of Gantt chart
```

```
df = []
df.append(dict(Task = process+str(proc[0]), Start = 0, Finish = bt[0]))
i = 1
finish = bt[0]
for i in range(1, len(bt)):
    df.append(dict(Task= process+str(proc[i]), Start=finish, Finish= finish +
bt[i]))
    finish = finish+bt[i]
fig = ff.create_gantt(df)
fig.show()

#FCFS Algorithm

def findWaitingTime(processes, n, bt, wt):
    wt[0] = 0
    for i in range(1, n ):
        wt[i] = bt[i - 1] + wt[i - 1]

def findTurnAroundTime(processes, n,  bt, wt, tat):
    for i in range(n):
        tat[i] = bt[i] + wt[i]

def findavgTime( processes, n, bt):

    wt = [0] * n
    tat = [0] * n
    total_wt = 0
    total_tat = 0

    findWaitingTime(processes, n, bt, wt)

    findTurnAroundTime(processes, n,  bt, wt, tat)

    print( "Processes Burst time " + " Waiting time " + " Turn around time")

    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" " + str(i + 1) + "\t\t" +
                    str(bt[i]) + "\t " +
                    str(wt[i]) + "\t\t " +
                    str(tat[i]))
    print( "Average waiting time = "+
                   str(total_wt / n))
    print("Average turn around time = "+
                     str(total_tat / n))
```

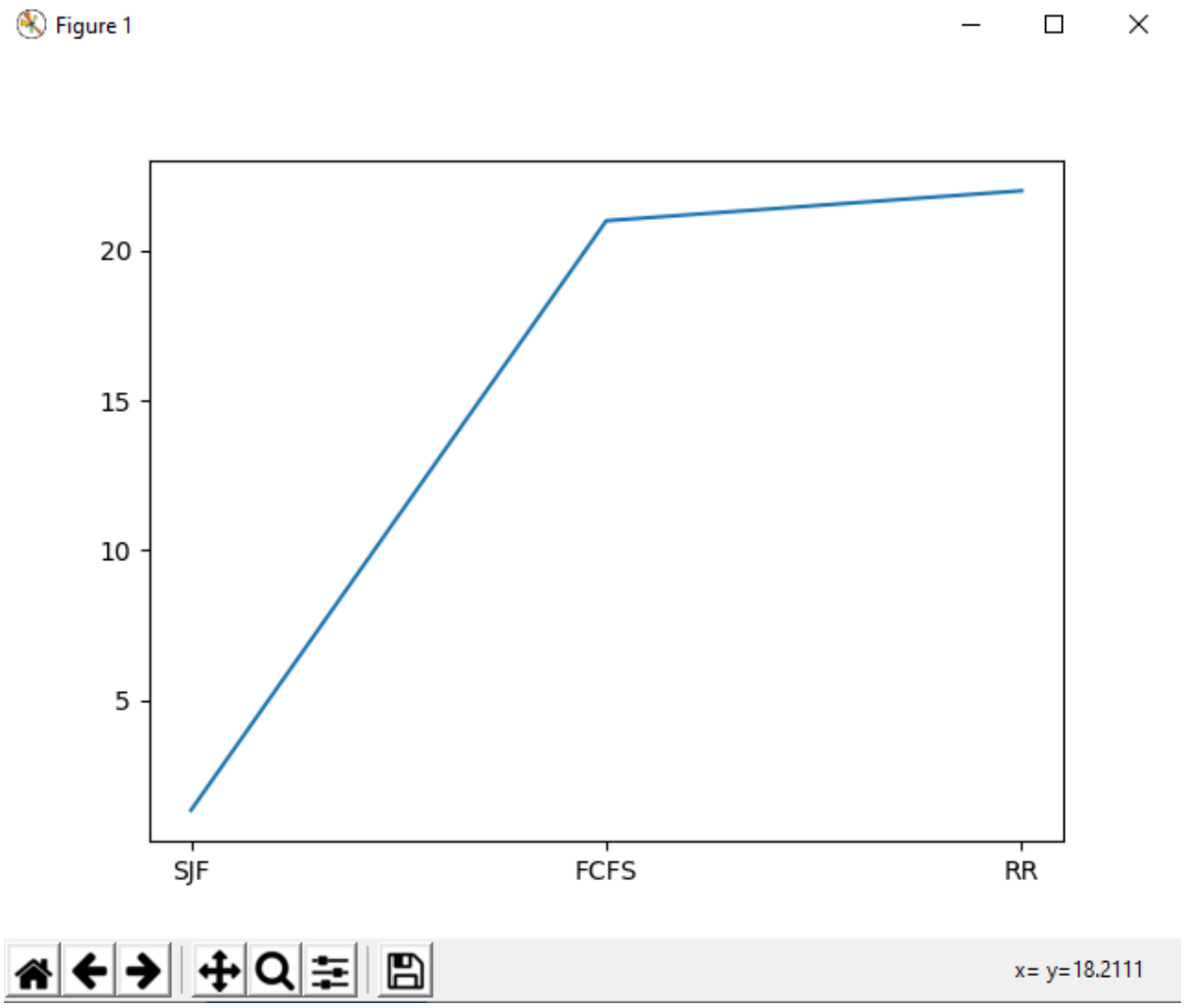
```
    return total_wt

# Driver code

if __name__ =="__main__":

    Dict = {}
    for i in range(1,sheet.nrows):
        Dict[sheet.cell_value(i,2)] = sheet.cell_value(i,0)
    print(Dict)
    processes1 = []
    processes1 = sorted(Dict.keys())
    processes = []
    for i in range(len(processes1)):
        processes.append(int(Dict[processes1[i]]))

    totalwait = findavgTime(processes, len(bt), bt)

    #Plot

    x = np.array([0,1,2])
    y = np.array([totalwait/len(bt),21,22])
    my_xticks = ['SJF','FCFS','RR']
    plt.xticks(x, my_xticks)
    plt.plot(x, y)
    plt.show()

    processes = []
    for i in range(1,sheet.nrows):
        processes.append(sheet.cell_value(i, 2))
    processes.sort()
```

INPUT FILE

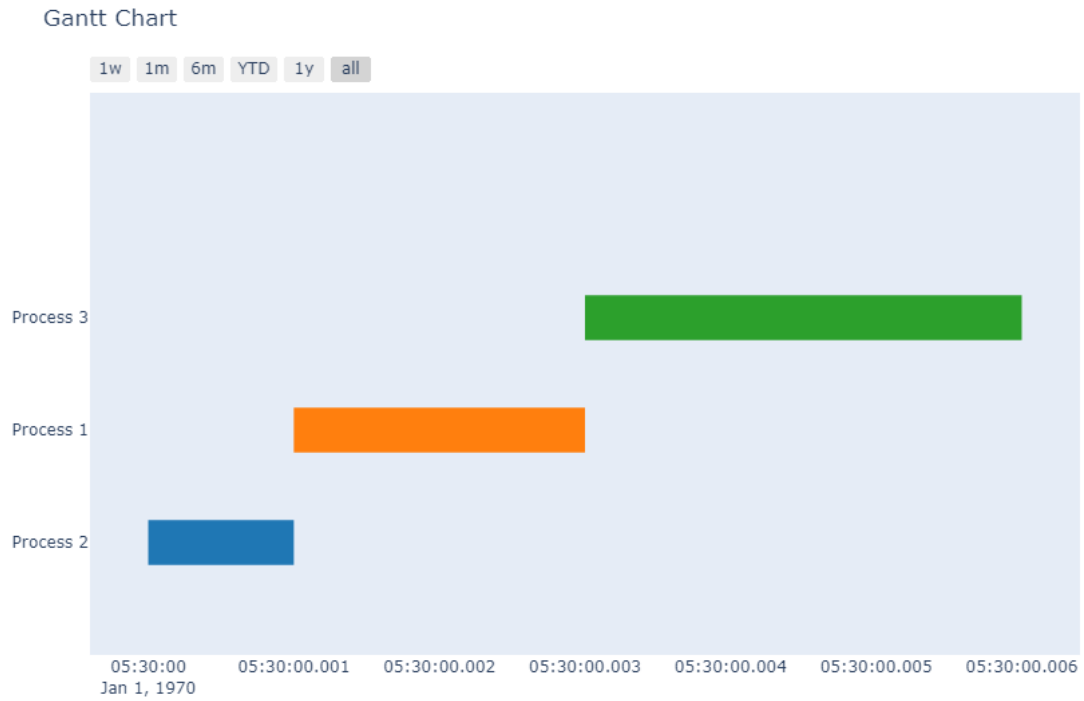
	A	B	C	D
1	Process	Duration	Arrival Time	
2	1	2	1	
3	2	1	2	
4	3	3	3	
5				

OUTPUT

```
Processes Burst time  Waiting time  Turn around time
 1              2        0              2
 2              1        2              3
 3              3        3              6
Average waiting time = 1.6666666666666667
Average turn around time = 3.6666666666666665
```



WAITING TIME GRAPH



b) SJF(Shortest Job First)

CODE

```
# Importing of libraries

import xlrd
import plotly.figure_factory as ff
import matplotlib.pyplot as plt
import numpy as np

process = "Process "

# Excel file linked

loc = ("Book1.xlsx")

wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)
sheet.cell_value(0, 0)

# Initializing burst_time and process list

Dict1 = {}
for i in range(1,sheet.nrows):
    Dict1[sheet.cell_value(i,1)] = sheet.cell_value(i,0)
```

```
processes1 = []
processes1 = sorted(Dict1.keys())
proc = []
for i in range(len(processes1)):
    proc.append(int(Dict1[processes1[i]]))

bt = []

for i in range(1, sheet.nrows):
    bt.append(sheet.cell_value(i,1))
bt.sort()

# Creation of Gantt chart

df = []
df.append(dict(Task = process+str(proc[0]), Start = 0, Finish = bt[0]))
i = 1
finish = bt[0]
for i in range(1, len(bt)):
    df.append(dict(Task= process+str(proc[i]), Start=finish, Finish= finish +
bt[i]))
    finish = finish+bt[i]
fig = ff.create_gantt(df)
fig.show()



# SJF Algorithm

def findWaitingTime(processes, n, wt):
    rt = [0] * n

    # Copy the burst time into rt[]
    for i in range(n):
        rt[i] = processes[i][1]
    complete = 0
    t = 0
    minm = 999999999
    short = 0
    check = False

    # Process until all processes gets
    # completed
    while (complete != n):

        # Find process with minimum remaining
        # time among the processes that
        # arrives till the current time`
        for j in range(n):
```

```
            if ((processes[j][2] <= t) and
                (rt[j] < minm) and rt[j] > 0):
                minm = rt[j]
                short = j
                check = True
        if (check == False):
            t += 1
            continue

        # Reduce remaining time by one
        rt[short] -= 1

        # Update minimum
        minm = rt[short]
        if (minm == 0):
            minm = 999999999

        # If a process gets completely
        # executed
        if (rt[short] == 0):

            # Increment complete
            complete += 1
            check = False

            # Find finish time of current
            # process
            fint = t + 1

            # Calculate waiting time
            wt[short] = (fint - proc1[short][1] -
                                proc1[short][2])

            if (wt[short] < 0):
                wt[short] = 0

        # Increment time
        t += 1

# Function to calculate turn around time
def findTurnAroundTime(processes, n, wt, tat):

    # Calculating turnaround time
    for i in range(n):
        tat[i] = processes[i][1] + wt[i]

# Function to calculate average waiting
# and turn-around times.
```

```
def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n

    # Function to find waiting time
    # of all processes
    findWaitingTime(processes, n, wt)

    # Function to find turn around time
    # for all processes
    findTurnAroundTime(processes, n, wt, tat)

    # Display processes along with all details
    print("Processes    Burst Time     Waiting",
                   "Time     Turn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", processes[i][0], "\t\t",
                   processes[i][1], "\t\t",
                   wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = ", total_tat / n)
    return total_wt

# Driver code
if __name__ =="__main__":

    #Making the table into suitable format for input

    proc1 = []

    for i in range(1,sheet.nrows):
        newlist=[]
        for j in range(0, sheet.ncols):
            newlist.append(int(sheet.cell_value(i,j)))
        proc1.append(newlist)

    n = len(bt)
    print(proc1)
    total_wait = findavgTime(proc1, n)

    #Plot
    x = np.array([0,1,2])
```

```
    y = np.array([total_wait/len(proc),21,22])
    my_xticks = ['SJF','FCFS','RR']
    plt.xticks(x, my_xticks)
    plt.plot(x, y)
    plt.show()
```

INPUT FILE

	A	B	C	D	E
1	Process	Duration	Arrival Time		
2	1	2	2		
3	2	1	1		
4	3	3	3		
5					
6					

OUTPUT

```
Processes    Burst Time     Waiting Time     Turn-Around Time
  1              2               0               2
  2              1               0               1
  3              3               1               4

Average waiting time = 0.33333
Average turn around time =  2.3333333333333335
```

c) Round Robin

```
# Importing of libraries

import xlrd
import plotly.figure_factory as ff
import matplotlib.pyplot as plt
import numpy as np

process = "Process "

# Excel file linked

loc = ("Book1.xlsx")

wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)
sheet.cell_value(0, 0)


def findWaitingTime(processes, n, bt,
                        wt, quantum):
    rem_bt = [0] * n

    # Copy the burst time into rt[]
    for i in range(n):
        rem_bt[i] = bt[i]
    t = 0 # Current time

    # Keep traversing processes in round
    # robin manner until all of them are
    # not done.
    while(1):
        done = True

        # Traverse all processes one by
        # one repeatedly
        for i in range(n):

            # If burst time of a process is greater
            # than 0 then only need to process further
            if (rem_bt[i] > 0) :
                done = False # There is a pending process

                if (rem_bt[i] > quantum) :

                    # Increase the value of t i.e. shows
                    # how much time a process has been processed
                    t += quantum
```

```
                    # Decrease the burst_time of current
                    # process by quantum
                    rem_bt[i] -= quantum

                # If burst time is smaller than or equal
                # to quantum. Last cycle for this process
                else:

                    # Increase the value of t i.e. shows
                    # how much time a process has been processed
                    t = t + rem_bt[i]

                    # Waiting time is current time minus
                    # time used by this process
                    wt[i] = t - bt[i]

                    # As the process gets fully executed
                    # make its remaining burst time = 0
                    rem_bt[i] = 0

        # If all processes are done
        if (done == True):
            break

# Function to calculate turn around time
def findTurnAroundTime(processes, n, bt, wt, tat):

    # Calculating turnaround time
    for i in range(n):
        tat[i] = bt[i] + wt[i]


# Function to calculate average waiting
# and turn-around times.
def findavgTime(processes, n, bt, quantum):
    wt = [0] * n
    tat = [0] * n

    # Function to find waiting time
    # of all processes
    findWaitingTime(processes, n, bt,
                        wt, quantum)

    # Function to find turn around time
    # for all processes
    findTurnAroundTime(processes, n, bt,
                                wt, tat)
```

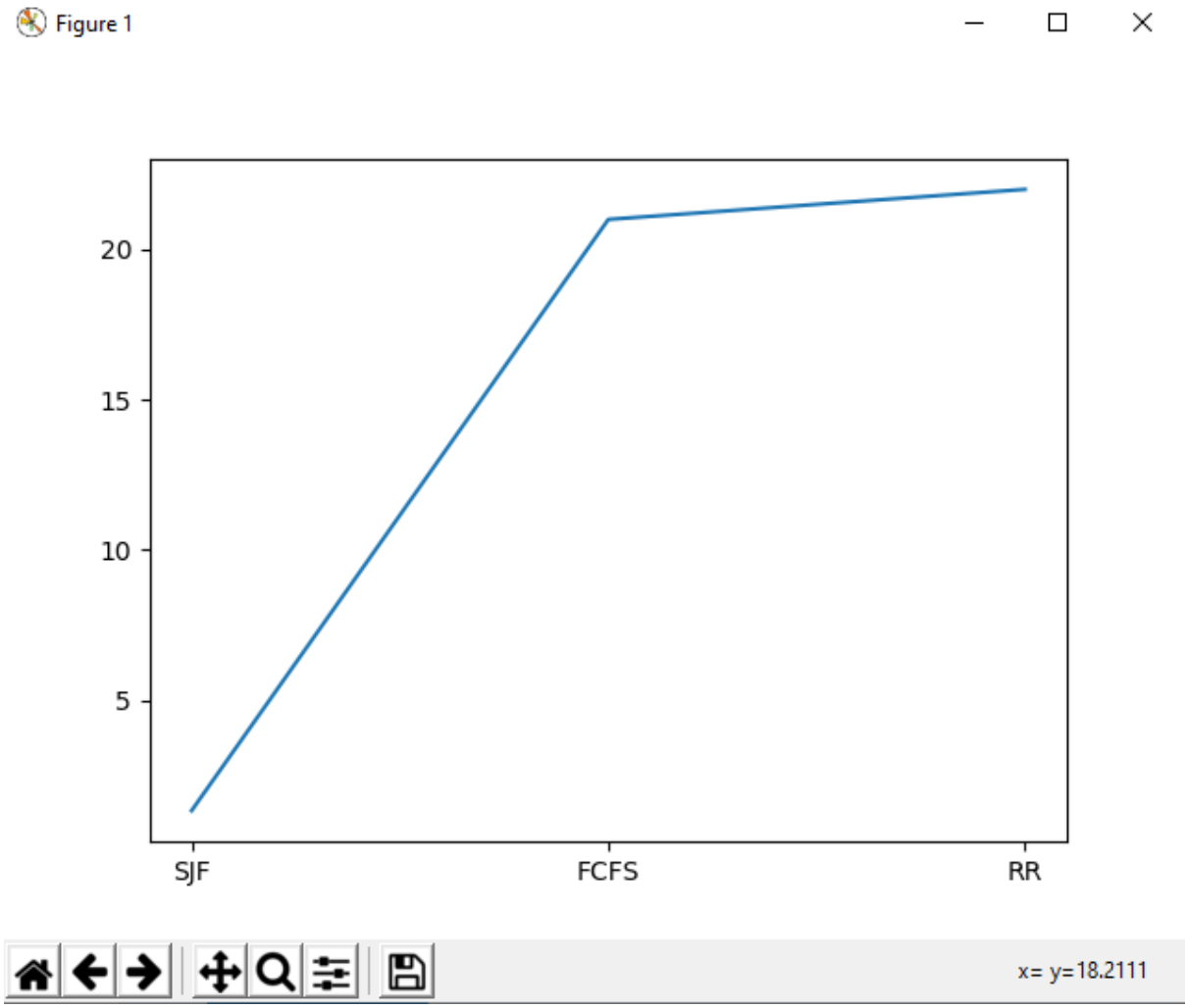
```
    # Display processes along with all details
    print("Processes    Burst Time     Waiting",
                    "Time    Turn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", i + 1, "\t\t", bt[i],
              "\t\t", wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = %.5f "% (total_tat / n))

# Driver code
if __name__ =="__main__":

    # Process id's
    proc = []
    for i in range(1,sheet.nrows):
        proc.append(sheet.cell_value(i, 0))
    n = len(proc)

    # Burst time of all processes
    bt = []
    for i in range(1,sheet.nrows):
        bt.append(sheet.cell_value(i, 1))

    # Time quantum
    quantum = 2
    findavgTime(proc, n, bt, quantum)
```

INPUT FILE

	A	B	C	D	E
1	Process	Duration	Arrival Time		
2	1	2	2		
3	2	1	1		
4	3	3	3		
5					
6					

OUTPUT

```
Processes    Burst Time     Waiting Time    Turn-Around Time
  1             2.0             0.0             2.0
  2             1.0             2.0             3.0
  3             3.0             3.0             6.0

Average waiting time = 1.66667
Average turn around time = 3.66667
```



WAITING TIME GRAPH

CONCLUSION

The project intends to simulate a CPU scheduler, which, selects a task to run from ready queue based on the scheduling algorithm first come first serve, shortest job first or round robin, so it
does not require any actual process creation or execution. When a task is scheduled, the simulator will simply print out what task is selected to run at a time. It outputs the way similar to Gantt chart style.
The **SJF scheduling algorithm** is better than FCFS as the problems like the convoy effect (if a process is very large, the processes which arrive after that process has to wait for a much longer time even if they have a job of only a few time units) does not occur in this type of schedule. However, the longer processes may be delayed for a much longer period of time in this type of schedule which is its major drawback.

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

1. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
2. One of the most commonly used technique in CPU scheduling as a core.
3. It is pre-emptive as processes are assigned CPU only for a fixed slice of time at most.
4. The disadvantage of it is more overhead of context switching.

REFERENCES

Working of FCFS
https://www.google.com/amp/s/www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/amp/

Working of SJF
https://www.studytonight.com/operating-system/shortest-job-first

Working of Round Robin
https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/

Python Gantt Chart library documentation:
https://plot.ly/python/gantt/

Python Graphs documentation:
https://www.geeksforgeeks.org/data-visualization-different-charts-python/