Kuwait University

College Engineering and Petroleum

Computer Engineering Department

CpE 0612445: Operating System Principles

Semester: Fall 2022-2023

Section No. 02

Project: CPU Scheduling Evaluation

Student Name: Yassine Nader Serrid
Student Id: 2181156439

Student Name: Dhari Al-Khaldi
Student Id: 2171119110

Instructor Name: Dr. Shouq Al-Subaihi

TA Name: Eng. Sarah Al-Swayed

Date: 12th Of December 20200

Table Of Contents

Table Of Figures

Introduction

The task of determining how and in what order to execute programs is known as CPU scheduling. A module called a dispatcher gives a process access to the CPU. There are six different types of process scheduling algorithms. It considerably affects the system's use of resources and general performance. The primary function of scheduling is to ensure that the operating system always has at least one process available in the ready queue whenever the CPU is idle. Deterministic modeling will be used to evaluate the performance of CPU scheduling and determine each algorithm's effectiveness for a specific workload.

Scheduling Algorithms

There are different criteria for selecting between different CPU scheduling algorithms. This project will use two types to demonstrate the scheduling strategy and how the processes are executed. Preemptive Shortest job first (SJF) and Non-Preemptive priority are the types implemented in this project. The following criterion of choosing the CPU scheduling algorithm is minimizing the average turnaround time and the average waiting time to increase throughput.

Shortest Job First SJF (Preemptive)

The Preemptive Shortest Job First (SJF) scheduling is particularly suitable for batch jobs whose run times are known in advance. For a given collection of processes, the Shortest Job First (SJF) algorithm provides the best average turnaround time, but it suffers from starvation for lengthy processes. The process with the shortest remaining time to completion is chosen to run in this scheduling method. Processes will always continue until they finish, or a new process is started that takes less time, as the currently running process is the one with the smallest amount of time left, and that time should only decrease as execution advances.

Priority (Non-Preemptive)

The processes are scheduled using a non-preemptive priority system that corresponds with the priority number given to them. The process will run till it is finished after it has been scheduled. The procedure generally has a greater priority when the priority number is smaller.

Implementation

The project will be designed and implemented using python programming language using classes concept to improve the program's flow and make it more appropriate to read while avoiding having a spaghetti code.

Process Generator.py

An empty list will be initialized to append processes following the constraints. The process generator python file will contain the main function, Generate Random Process, which will generate and append the process ID values, Arrival time, CPU burst, and priority. The first appended process will have a default arrival time which is zero associated with the process ID.

The first for loop will be responsible for appending the process with the selected number of processes by the user following the required constraints that the priority will not be greater than 20 with CPU burst between 0 to 30. The second for loop will be used to append arrival time starting from the second process. The Third for loop will be used to append processes with the exact arrival time with a constraint that the number of processes with the same arrival time doesn't exceed five. After generating the process and appending all the processes, the values will be added to the text file as required.

Table.py

Python file here contains one function responsible for appending the list generated from the process generator file in the text file. First, a text file should be added to the python project's folder. The function will open the text file to write the values following the text file provided in the project's description.

SJF.py

This python file will be responsible for simulating Preemptive SJF among a selected number of processes the user selects in the main program, as explained in the main program explanation. The run function will contain the steps of the SJF algorithm. First, an empty list called Gantt will be created to append the processes execution flow. The processes parameter (A list created in the main) will be sorted according to the process's arrival time. Another empty list will be created with the burst time of the processes. The first process will be appended as it has an arrival time of 0 with a response time of 0. A counter variable with time will be created to keep track of the processes execution flow. A while loop will be used to simulate the processes to stop in conditions if the process's burst time stops if the list becomes all zeros.

Priority.py

This python file will be responsible for simulating Non-Preemptive Priority among a selected number of processes the user selects in the main program, as explained in the main program explanation. First, an empty list will be initialized called Gantt to append the process flow. We initialized variables that will be updated in each iteration. Processes will be sorted according to their priority first; after sorting, the processes will be according to the arrival time. The first processes from the proc list will be appended first in the Gantt list since the priority and the arrival time sort it; even if it has the lowest priority, it will be appended since its arrival time is the lower. For loop and the nested for loop will be used to simulate the processes.

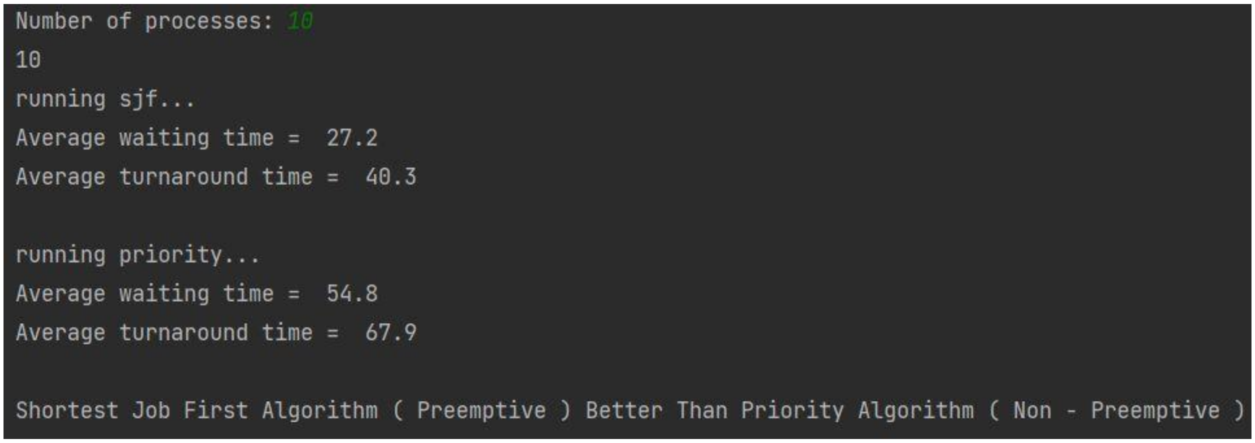
Main.py

The main program will be in charge of combining the codes above by importing the previous files. A Process class will be created to initialize the variables for creating the processes as objects in generating processes step. An empty list will be initialized and used as a parameter for the called functions of the CPU scheduling algorithms. A function called read process will be used to open the text file, read the processes information line by line, and append it to the empty list created. In the main function, the user will be asked to enter the number of processes to simulate the CPU scheduling algorithms and evaluate which algorithm provides a better average waiting time and turnaround time. Processes will not be changed after it has been generated randomly from the process generator python file.

Test Cases

To start testing both CPU scheduling algorithms first we need to generate a list of process generator python file, here we will generate 1000 processes following the constraints, in the main program the user will specify the number of processes desired from that text file.

Test Case 1: 10 Processes:

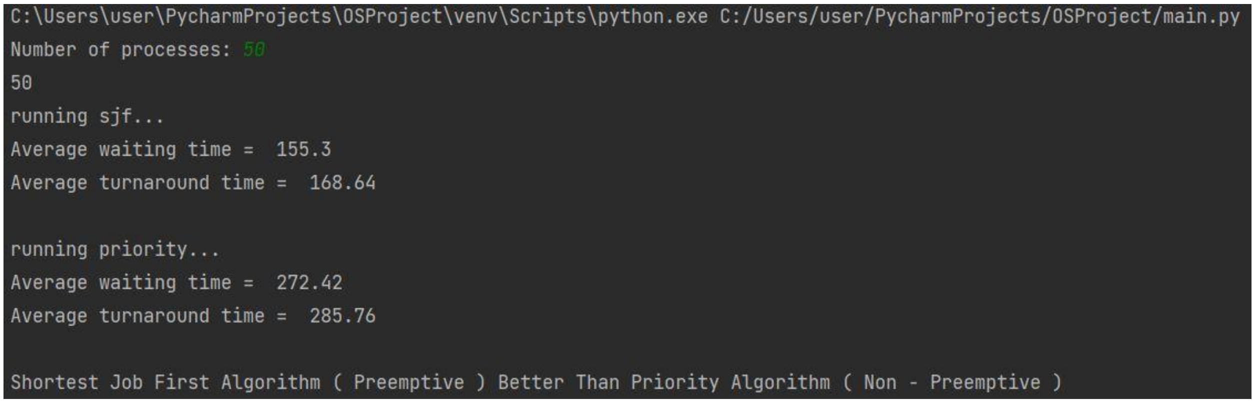


```
Number of processes: 10
10
running sjf...
Average waiting time =  27.2
Average turnaround time =  40.3

running priority...
Average waiting time =  54.8
Average turnaround time =  67.9

Shortest Job First Algorithm ( Preemptive ) Better Than Priority Algorithm ( Non - Preemptive )
```

Figure 1. Test Case 1: 10 Processes.

In Figure 1, shows the results of Average waiting time and turnaround time applied on 10 processes. After simulating both algorithms, we can notice that the shortest job first is better than priority algorithm in both criteria.

Test Case 2: 50 Processes:

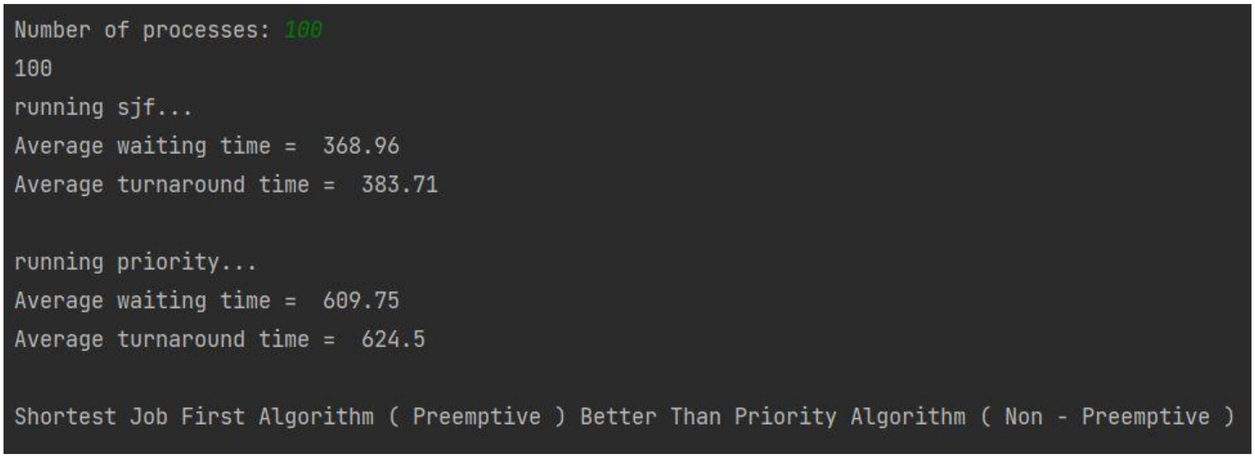


```
C:\Users\user\PycharmProjects\OSProject\venv\Scripts\python.exe C:/Users/user/PycharmProjects/OSProject/main.py
Number of processes: 50
50
running sjf...
Average waiting time =  155.3
Average turnaround time =  168.64

running priority...
Average waiting time =  272.42
Average turnaround time =  285.76

Shortest Job First Algorithm ( Preemptive ) Better Than Priority Algorithm ( Non - Preemptive )
```

Figure 2. Test Case 1: 50 Processes.

In Figure 2, shows the results of Average waiting time and turnaround time applied on 50 processes. After simulating both algorithms, we can notice that the shortest job first is better than priority algorithm in both criteria.

Test Case 3: 100 Processes:

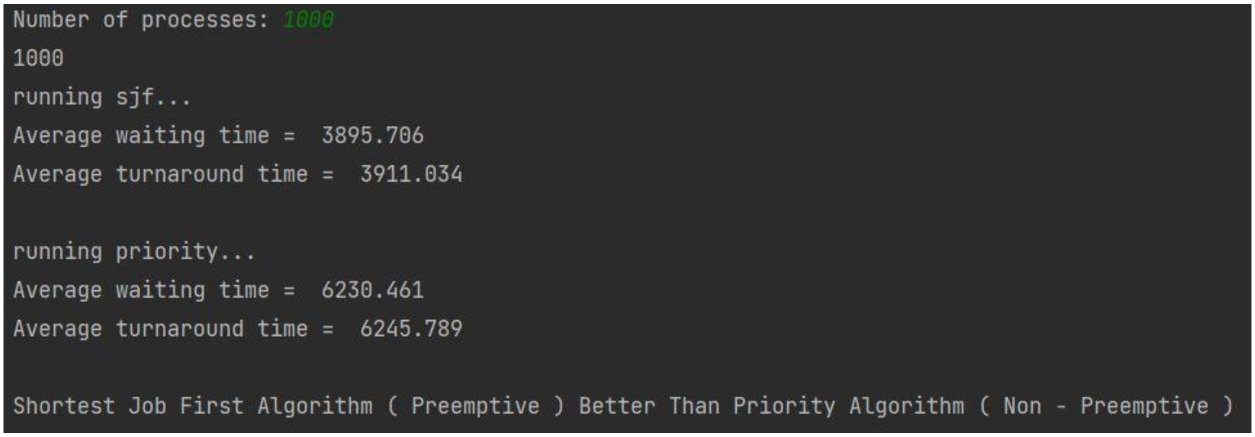


```
Number of processes: 100
100
running sjf...
Average waiting time =  368.96
Average turnaround time =  383.71

running priority...
Average waiting time =  609.75
Average turnaround time =  624.5

Shortest Job First Algorithm ( Preemptive ) Better Than Priority Algorithm ( Non - Preemptive )
```

Figure 3. Test Case 1: 100 Processes.

In Figure 3, shows the results of Average waiting time and turnaround time applied on 100 processes. After simulating both algorithms, we can notice that the shortest job first is better than priority algorithm in both criteria.

Test Case 4: 1000 Processes:



```
Number of processes: 1000
1000
running sjf...
Average waiting time =  3895.706
Average turnaround time =  3911.034

running priority...
Average waiting time =  6230.461
Average turnaround time =  6245.789

Shortest Job First Algorithm ( Preemptive ) Better Than Priority Algorithm ( Non - Preemptive )
```

Figure 4. Test Case 1: 1000 Processes.

In Figure 4, shows the results of Average waiting time and turnaround time applied on 1000 processes. After simulating both algorithms, we can notice that the shortest job first is better than priority algorithm in both criteria.

Conclusion

As discussed in each algorithm, the best algorithm for the lowest average waiting time and average turnaround time. Additionally, compared to personal calculations, this project saves a significant amount of time when estimating the average wait time for a workload. In this project we can notice that the SJF algorithm is more effective than the priority algorithm in all cases in both average waiting time and turnaround time in most cases as examined.

Reference

[1]. Algorithm Evaluation for Scheduling. (2022). Retrieved 12 December 2022, from https://mycareerwise.com/content/algorithm-evaluation-for-scheduling/content/exam/gate/computer-science

[2]. (PDF) A comparative study of CPU scheduling algorithms (no date). Available at: https://www.researchgate.net/publication/249645533_A_Comparative_Study_of_CPU_Scheduling_Algorithms (Accessed: December 12, 2022).

[3]. Program for Shortest Job First (SJF) scheduling | Set 2 (Preemptive) - Tutorialspoint.dev - TutorialsPoint.dev. (2022). Retrieved 12 December 2022, from https://tutorialspoint.dev/computer-science/operating-systems/program-shortest-job-first-scheduling-set-2srtf-make-changesdoneplease-review

Appendices

Appendix A: Meating Minutes

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #1

Team Name:	**OS TEAM**	**Date:**	**15/11/2022**
Start Time:	**7:00pm**	**Finish Time:**	**9:45pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Discussing in which language to program in
- Searching for algoritms and understanding the methods

Task list:

Team Member	**Assigned tasks**	**Start Date**	**Due Date**
Yassine	**Start working on source code**	**15/11/2022**	**16/11/2022**
Dhari	**Downloading PyCharm and starts learn basics**	**15/11/2022**	**16/11/2022**

Next meeting will be held on (16/11/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #2

Team Name:	**OS TEAM**	**Date:**	**16/11/2022**
Start Time:	**7:30pm**	**Finish Time:**	**11:00pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Start working on the code

Task list:

Team Member	Assigned tasks	Start Date	Due Date
Yassine	**Continue working on the code**	**17/11/2022**	**20/11/2022**
Dhari	**Assist Yassine on the code**	**17/11/2022**	**20/11/2022**

Next meeting will be held on (21/11/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #3

Team Name:	**OS TEAM**	**Date:**	**21/11/2022**
Start Time:	**8:00pm**	**Finish Time:**	**10:45pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Searching for codes and methods

Task list:

Team Member	Assigned tasks	Start Date	Due Date
Yassine	**Continue working on the code**	**21/11/2022**	**25/11/2022**
Dhari	**Continue searching for helpful source codes**	**21/11/2022**	**25/11/2022**

Next meeting will be held on (26/11/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #3

Team Name:	**OS TEAM**	**Date:**	**26/11/2022**
Start Time:	**4:00pm**	**Finish Time:**	**8:00pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Working on the functions

Task list:

Team Member	**Assigned tasks**	**Start Date**	**Due Date**
Yassine	**Working on the code**	**26/11/2022**	**30/11/2022**
Dhari	**Working on the code**	**26/11/2022**	**30/11/2022**

Next meeting will be held on (30/11/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #4

Team Name:	**OS TEAM**	**Date:**	**30/11/2022**
Start Time:	**6:00pm**	**Finish Time:**	**9:00pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Fixing errors and modify functions

Task list:

Team Member	**Assigned tasks**	**Start Date**	**Due Date**
Yassine	**Fixing errors / Report**	**31/11/2022**	**4/12/2022**
Dhari	**Work on functions**	**31/11/2022**	**4/12/2022**

Next meeting will be held on (4/12/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #5

Team Name:	**OS TEAM**	**Date:**	**4/12/2022**
Start Time:	**9:00pm**	**Finish Time:**	**12:00pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Working on completion of the code

Task list:

Team Member	**Assigned tasks**	**Start Date**	**Due Date**
Yassine	**Fixing errors**	**4/12/2022**	**6/12/2022**
Dhari	**Report**	**4/12/2022**	**6/12/2022**

Next meeting will be held on (7/12/2022)

Kuwait University

College of Engineering and Petroleum
Computer Engineering Department

Team Meeting Minutes #6

Team Name:	**OS TEAM**	**Date:**	**10/12/2022**
Start Time:	**5:00pm**	**Finish Time:**	**10:00pm**

Members Present:	**Dhari-Yassine**
Members Excused:	**-**
Members Tardy:	**-**
Members Absent:	**-**

Summary of meeting

- Testing and fixing

Task list:

Team Member	Assigned tasks	Start Date	Due Date
Yassine	**Fixing code**	**10/12/2022**	**12/12/2022**
Dhari	**Report/fixing code**	**10/12/2022**	**12/12/2022**

Appendix B: Source Code

Table.py

```python
def ProcessInfo(processes):
    fp = open("processes.txt", "w")

    x = len(processes)
    fp.write(str(x) + '\n')
    fp.write(' ID       Arrival     CPU burst     Priority \n')
    for p in processes:
        fp.write("{:3}        {:3}         {:4}        {:4}
\n".format(p.p_id, p.arrival_time, p.burst_time, p.priority))

    fp.close()
```

ProcessGenerator.py

```
import random
import Table as table
from main import Process

processes = []

def generateRandomProcess(n, burst, priority):
    processes.append(Process(0+1, 0, random.randint(1, burst),
random.randint(1, priority)))
    for i in range(1, n):
        p = Process(i+1, 0, random.randint(1, burst), random.randint(1,
priority))
        processes.append(p)

    for i in range(1, n):
        at = processes[i-1].arrival_time + random.randint(1, 5)
        processes[i].arrival_time = at

    #constraint in the arrival_time
    x = random.randint(0, n/2) # x + 1 the repeated values will appear
    y = random.randint(2, 5) # the number of repeated values that will
appear

    for i in range(0, y):
        processes[x+i].arrival_time = processes[x].arrival_time

    table.ProcessInfo(processes)

def main(n, burst, priority):
    generateRandomProcess(n, burst, priority)

if __name__ == '__main__':
    priority = 20
    burst = 30
    n = int(input("Number of processes: "))
    main(n, burst, priority)
```

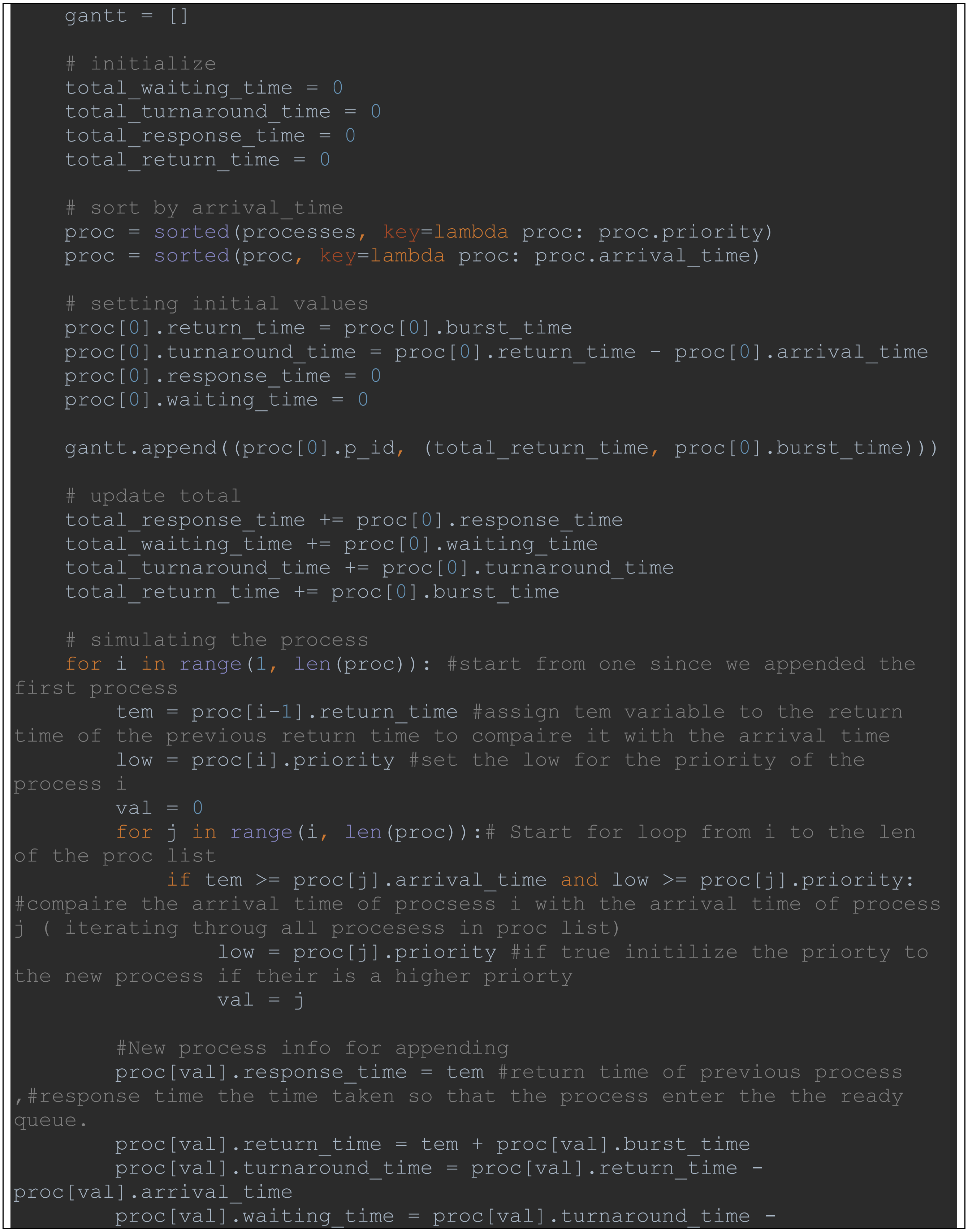
SJF.py

```
import Table as table

def run(processes):

    print('running sjf...')
    gantt = []

    # initialize
    total_waiting_time = 0
    total_turnaround_time = 0
    total_response_time = 0
    total_return_time = 0

    # sort by arrival_time
    proc = sorted(processes, key=lambda proc: proc.arrival_time)

    burst_time_list = []
    for i in range(len(proc)):
        burst_time_list.append(proc[i].burst_time)

    gantt.append([proc[0].p_id, [0, 0]])
    proc[0].response_time = 0

    time = 0
    minimum = 999999999999          # Initiating minimum burst time
    curr_indx = 0
    running = False # there is process or no

    while burst_time_list != [0] * len(proc):

        for i in range(len(burst_time_list)): # check for minimum burst 
time
            if proc[i].arrival_time <= time and minimum > 
burst_time_list[i] > 0:
                minimum = burst_time_list[i] #value of the minimum cpu 
burst
                curr_indx = i #index of the minimum CPU burst
                running = True # if their is a minimum it will work

        if running is False: # their no process working
            time += 1 #increment the timer
            continue # go back to the while loop

        if curr_indx != gantt[-1][0] - 1:  #check current index and the 
the running current index         # The process running changed ( if the 
curr_indx doestn't equal to the last process in the gantt)
            gantt.append([proc[curr_indx].p_id, [time, 0]]) # append new 
tuple (process in the gantt

        gantt[-1][1][1] += 1 #increment burst time which running time
        burst_time_list[curr_indx] -= 1 #decrement the CPU burst ( 
remaining burst time)

        minimum = burst_time_list[curr_indx] #intialize the the new 
minimum value affter decremting the CPU burst go back to the for loop and 
compare it
```

```
        if minimum == 0:                # A processes is completely
executed
            minimum = 999999999999      # Reset minimum burst time

            running = False

            proc[curr_indx].return_time = time + 1
            total_return_time += proc[curr_indx].burst_time

            proc[curr_indx].waiting_time = proc[curr_indx].return_time -
proc[curr_indx].burst_time - proc[curr_indx].arrival_time #formula

            if proc[curr_indx].waiting_time < 0:
                proc[curr_indx].waiting_time = 0

            total_waiting_time += proc[curr_indx].waiting_time

        time += 1
    for i in range(len(gantt)):
        gantt[i][1] = tuple(gantt[i][1])
        gantt[i] = tuple(gantt[i])


    for i in range(len(proc)):
        proc[i].turnaround_time = proc[i].burst_time +
proc[i].waiting_time
        total_turnaround_time += proc[i].turnaround_time


    return {
        'name': 'PR-NP',
        'avg_waiting_time': total_waiting_time / len(proc),
        'avg_response_time': total_response_time / len(proc),
        'avg_turnaround_time': total_turnaround_time / len(proc),
        'processes': proc,
        'gantt': gantt
    }
```

Priority.py

```
import Table as table

def run(processes):

    print('running priority...')
```

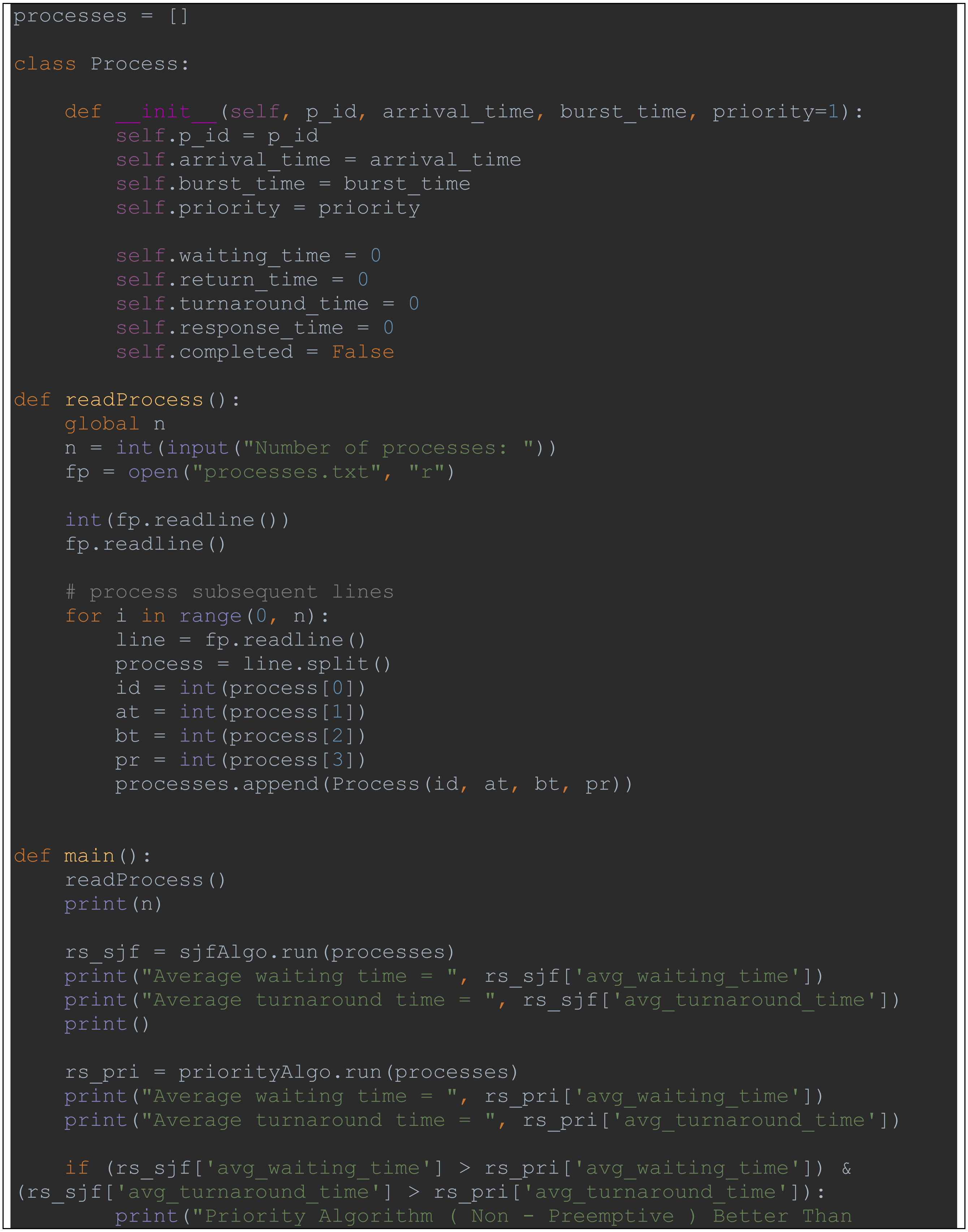


```
    gantt = []

    # initialize
    total_waiting_time = 0
    total_turnaround_time = 0
    total_response_time = 0
    total_return_time = 0

    # sort by arrival_time
    proc = sorted(processes, key=lambda proc: proc.priority)
    proc = sorted(proc, key=lambda proc: proc.arrival_time)

    # setting initial values
    proc[0].return_time = proc[0].burst_time
    proc[0].turnaround_time = proc[0].return_time - proc[0].arrival_time
    proc[0].response_time = 0
    proc[0].waiting_time = 0

    gantt.append((proc[0].p_id, (total_return_time, proc[0].burst_time)))

    # update total
    total_response_time += proc[0].response_time
    total_waiting_time += proc[0].waiting_time
    total_turnaround_time += proc[0].turnaround_time
    total_return_time += proc[0].burst_time

    # simulating the process
    for i in range(1, len(proc)): #start from one since we appended the
first process
        tem = proc[i-1].return_time #assign tem variable to the return
time of the previous return time to compaire it with the arrival time
        low = proc[i].priority #set the low for the priority of the
process i
        val = 0
        for j in range(i, len(proc)):# Start for loop from i to the len
of the proc list
            if tem >= proc[j].arrival_time and low >= proc[j].priority:
#compaire the arrival time of procsess i with the arrival time of process
j ( iterating throug all processess in proc list)
                low = proc[j].priority #if true initilize the priorty to
the new process if their is a higher priorty
                val = j

        #New process info for appending
        proc[val].response_time = tem #return time of previous process
,#response time the time taken so that the process enter the the ready
queue.
        proc[val].return_time = tem + proc[val].burst_time
        proc[val].turnaround_time = proc[val].return_time -
proc[val].arrival_time
        proc[val].waiting_time = proc[val].turnaround_time -
```

```
proc[val].burst_time

        gantt.append(
            (proc[val].p_id, (total_return_time, proc[val].burst_time)))
#appending the new process

        proc[i], proc[val] = proc[val], proc[i] #swapping

        # update total
        total_response_time += proc[i].response_time
        total_waiting_time += proc[i].waiting_time
        total_turnaround_time += proc[i].turnaround_time
        total_return_time += proc[i].burst_time

    proc = sorted(proc, key=lambda proc: proc.arrival_time)

    return {
        'name': 'PR-NP',
        'avg_waiting_time': total_waiting_time/len(proc),
        'avg_response_time': total_response_time/len(proc),
        'avg_turnaround_time': total_turnaround_time/len(proc),
        'processes': proc,
        'gantt': gantt
    }
```

main.py

```
import SJF as sjfAlgo
import Priority as priorityAlgo
import Table as table
```



```
processes = []

class Process:

    def __init__(self, p_id, arrival_time, burst_time, priority=1):
        self.p_id = p_id
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.priority = priority

        self.waiting_time = 0
        self.return_time = 0
        self.turnaround_time = 0
        self.response_time = 0
        self.completed = False

def readProcess():
    global n
    n = int(input("Number of processes: "))
    fp = open("processes.txt", "r")

    int(fp.readline())
    fp.readline()

    # process subsequent lines
    for i in range(0, n):
        line = fp.readline()
        process = line.split()
        id = int(process[0])
        at = int(process[1])
        bt = int(process[2])
        pr = int(process[3])
        processes.append(Process(id, at, bt, pr))


def main():
    readProcess()
    print(n)

    rs_sjf = sjfAlgo.run(processes)
    print("Average waiting time = ", rs_sjf['avg_waiting_time'])
    print("Average turnaround time = ", rs_sjf['avg_turnaround_time'])
    print()

    rs_pri = priorityAlgo.run(processes)
    print("Average waiting time = ", rs_pri['avg_waiting_time'])
    print("Average turnaround time = ", rs_pri['avg_turnaround_time'])

    if (rs_sjf['avg_waiting_time'] > rs_pri['avg_waiting_time']) &
(rs_sjf['avg_turnaround_time'] > rs_pri['avg_turnaround_time']):
        print("Priority Algorithm ( Non - Preemptive ) Better Than
Shortest Job First Algorithm ( Preemptive )")
    elif (rs_sjf['avg_waiting_time'] < rs_pri['avg_waiting_time']) &
(rs_sjf['avg_turnaround_time'] < rs_pri['avg_turnaround_time']):
        print("\nShortest Job First Algorithm ( Preemptive ) Better Than
Priority Algorithm ( Non - Preemptive )")
    elif (rs_sjf['avg_waiting_time'] < rs_pri['avg_waiting_time']) &
(rs_sjf['avg_turnaround_time'] > rs_pri['avg_turnaround_time']):
        print("\nShortest Job First Algorithm ( Preemptive ) Better In
The Average Waiting Time While Priority Algorithm ( Non - Preemptive )
Better In Average Turn Arount Time")
    elif (rs_sjf['avg_waiting_time'] > rs_pri['avg_waiting_time']) &
(rs_sjf['avg_turnaround_time'] < rs_pri['avg_turnaround_time']):
        print("\nShortest Job First Algorithm ( Preemptive ) Better In
The Average Turnaround Time While Priority Algorithm ( Non - Preemptive )
Better In Average Waiting Time")




if __name__ == '__main__':
    main()
```