CERTIFICATE

Name of the Lab : OPERATING SYSTEMS

Name of the Student : shaik.sharmila

Student Regd.No. : 19BQ5A0528

CLASS : III B.TECH. I SEM CSE – D

GITHUBlink:<https://github.com/sharmila528/os-lab>

19BQ5A0528Page 1

INDEX

S. No. Name of the Experiment Page No. 1 (a) Simulate FCFS CPU scheduling algorithm 3 1 (b) Simulate Non Preemptive SJF CPU scheduling algorithm 8 1 (c) Simulate Preemptive SJF CPU scheduling algorithm 14 1 (d) Simulate Non Preemptive Priority CPU scheduling algorithm 21 1 (e) Simulate Preemptive Priority CPU scheduling algorithm 29 1 (f) Simulate RoundRobin CPU scheduling algorithm 37 2 (a) Design a c program to implement the multiprogramming memory management implementation of Fork() using System call.

2 (b) Design a c program to implement the multiprogramming memory management implementation of exit() using System call.

2 (c) Design a c program to implement the multiprogramming memory management implementation of exec() using System call.

2 (d) Design a c program to implement the multiprogramming memory management implementation of wait() using System call.

19BQ5A0528Page 2

**EXPERIMENT NO: 1 (a)**

**AIM :** To implement program for FCFS scheduling Algorithm

**DESCRIPTION :** FCFS means First Come First Serve. That means , based on arrival times of the processes into the ready queue, the CPU schedules the processes. By default , the procedure of FCFS shows that FCFS is a non - preemptive scheduling algorithm. Because, here there is no interruption while one process is executing .Here a process can execute until it’s burst time .Then only CPU schedules for another process.

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

class FCFS:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class FCFS:

def processData(self, no\_of\_processes):

process\_data = []

for i in range(1,no\_of\_processes+1):

temporary = []

arrival\_time = int(input("Enter Arrival Time of Process {}:

".format(i)))

19BQ5A0528 Page 3

burst\_time = int(input("Enter Burst Time of Process {}: ".format(i)))

temporary.extend([i, arrival\_time, burst\_time])

process\_data.append(temporary)

FCFS.schedulingProcess(self, process\_data)

def schedulingProcess(self, process\_data):

start\_time = []

exit\_time = []

s\_time = 0

for i in range(len(process\_data)):

if(s\_time < process\_data[i][1]):

s\_time = process\_data[i][1]

start\_time.append(s\_time)

s\_time = s\_time + process\_data[i][2]

e\_time = s\_time

exit\_time.append(e\_time)

process\_data[i].append(e\_time)

t\_time = FCFS.calculateTurnaroundTime(self, process\_data)

w\_time = FCFS.calculateWaitingTime(self, process\_data)

FCFS.printData(self, process\_data, t\_time, w\_time)

def calculateTurnaroundTime(self, process\_data):

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][3] - process\_data[i][1]

total\_turnaround\_time += turnaround\_time

19BQ5A0528 Page 4

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time /

len(process\_data)

return average\_turnaround\_time

def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

waiting\_time = process\_data[i][4] - process\_data[i][2]

total\_waiting\_time = total\_waiting\_time + waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time,

average\_waiting\_time):

print("Process\_ID Arrival\_Time Burst\_Time Completion\_Time

Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

print('Average Waiting Time:',average\_waiting\_time)

19BQ5A0528 Page 5



print("Number of context switches:",no\_of\_processes-1)

no\_of\_processes = int(input("Enter number of processes: "))

fcfs = FCFS()

fcfs.processData(no\_of\_processes)

**OUTPUT:**

Enter number of processes: 3

Enter Arrival Time of Process 1: 0

Enter Burst Time of Process 1: 5

Enter Arrival Time of Process 2: 2

Enter Burst Time of Process 2: 3

Enter Arrival Time of Process 3: 4

Enter Burst Time of Process 3: 4

Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time 1 0 5 5 5 0

2 2 3 8 6 3 3 4 4 12 8 4

Average Turnaround Time: 6.333333333333333

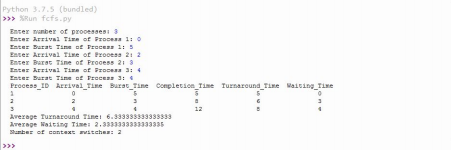
Average Waiting Time: 2.3333333333333335

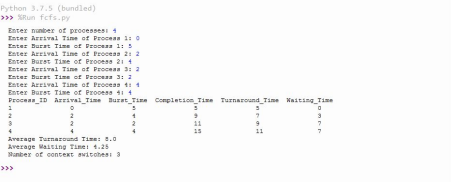
Number of context switches: 2

19BQ5A0528 Page 6

**OUTPUT SCREEN SHOTS:**

Output:1

Output:2

19BQ5A0528 Page 7

**EXPERIMENT NO: 1 (b)**

**AIM :** To implement program for Non Preemptive SJF scheduling Algorithm

**DESCRIPTION :** Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next.

Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.In this non-preemptive sjf,there is no interruption for the process in execution until its burst time.

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

class SJF:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class SJF:

def processData(self, no\_of\_processes):

process\_data = []

for i in range(1,no\_of\_processes+1):

temporary = []

arrival\_time = int(input("Enter Arrival Time for Process {}: ".format(i)))

burst\_time = int(input("Enter Burst Time for Process {}: ".format(i)))

19BQ5A0528 Page 8

temporary.extend([i, arrival\_time, burst\_time, 0])

process\_data.append(temporary)

SJF.schedulingProcess(self, process\_data)

def schedulingProcess(self, process\_data):

start\_time = []

exit\_time = []

s\_time = 0

process\_data.sort(key=lambda x: x[1])

for i in range(len(process\_data)):

ready\_queue = []

temp = []

normal\_queue = []

for j in range(len(process\_data)):

if ((process\_data[j][1] <= s\_time) and (process\_data[j][3] == 0)):

temp.extend([process\_data[j][0], process\_data[j][1],

process\_data[j][2]])

ready\_queue.append(temp)

temp = []

elif (process\_data[j][3] == 0):

temp.extend([process\_data[j][0], process\_data[j][1],

process\_data[j][2]])

normal\_queue.append(temp)

temp = []

19BQ5A0528 Page 9



if( len(ready\_queue) != 0):

ready\_queue.sort(key=lambda x: x[2])

start\_time.append(s\_time)

s\_time = s\_time + ready\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

for k in range(len(process\_data)):

if (process\_data[k][0] == ready\_queue[0][0]):

break

process\_data[k][3] = 1

process\_data[k].append(e\_time)

elif (len(ready\_queue) == 0):

if( s\_time < normal\_queue[0][1]):

s\_time = normal\_queue[0][1]

start\_time.append(s\_time)

s\_time = s\_time + normal\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

for k in range(len(process\_data)):

if (process\_data[k][0] == normal\_queue[0][0]):

break

process\_data[k][3] = 1

process\_data[k].append(e\_time)

19BQ5A0528 Page 10

t\_time = SJF.calculateTurnaroundTime(self, process\_data)

w\_time = SJF.calculateWaitingTime(self, process\_data)

SJF.printData(self, process\_data, t\_time, w\_time)

def calculateTurnaroundTime(self, process\_data):

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][4] - process\_data[i][1]

total\_turnaround\_time += turnaround\_time

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time / len(process\_data)

return average\_turnaround\_time

def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

waiting\_time = process\_data[i][5] - process\_data[i][2]

total\_waiting\_time = total\_waiting\_time + waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time):

print("Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

if(j==3):

19BQ5A0528 Page 11

continue

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

print('Average Waiting Time:',average\_waiting\_time)

print("Number of Context Switches:",no\_of\_processes-1)

no\_of\_processes = int(input("Enter number of processes: "))

sjf = SJF()

sjf.processData(no\_of\_processes)

**OUTPUT:**

Enter number of processes: 4

Enter Arrival Time for Process 1: 0

Enter Burst Time for Process 1: 4

Enter Arrival Time for Process 2: 3

Enter Burst Time for Process 2: 2

Enter Arrival Time for Process 3: 3

Enter Burst Time for Process 3: 4

Enter Arrival Time for Process 4: 5

Enter Burst Time for Process 4: 3

Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time 1 0 4 4 4 0 2 3 2 6 3 1

19BQ5A0528 Page 12

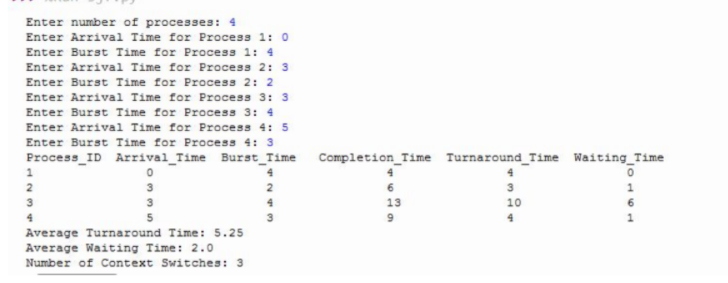
3 3 4 13 10 6 4 5 3 9 4 1 Average Turnaround Time: 5.25

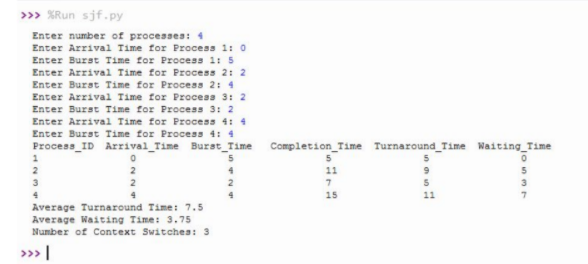
Average Waiting Time: 2.0

Number of Context Switches: 3

**OUTPUT SCREEN SHOTS:**

Output:1

Output:2

19BQ5A0528 Page 13

**EXPERIMENT NO: 1 (c)**

**AIM :** To implement program for Preemptive sjf scheduling Algorithm

**DESCRIPTION :** SJF: Shortest Job First.

The task is to find the Average Waiting Time and Average Turnaround Time of the given processes with their Burst Time using SJF Scheduling Algorithm.SJF is a scheduling policy that selects the waiting process with the smallest execution time to execute next.

Priority Scheduling is a Non Preemptive and Preemptive Algorithm, hence the process which has the Least Burst Time is selected first.Here we are considering Preemptive version of Priority Scheduling, hence the process which has the Least Burst Time will be served first and will be continued to be served till there is any other process with Lower Burst Time priority.if there is any process with Lower Burst Time, then switch the process.

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

class SRTF:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class Srtf:

def processData(self, no\_of\_processes):

process\_data = []

for i in range(1,no\_of\_processes+1):

temporary = []

19BQ5A0528 Page 14

arrival\_time = int(input("Enter Arrival Time for Process {}: ".format(i)))

burst\_time = int(input("Enter Burst Time for Process {}: ".format(i)))

temporary.extend([i, arrival\_time, burst\_time, 0, burst\_time])

process\_data.append(temporary)

Srtf.schedulingProcess(self, process\_data)

def schedulingProcess(self, process\_data):

start\_time = []

exit\_time = []

s\_time=0

sequence\_of\_process = []

process\_data.sort(key=lambda x: x[1])

while( 1):

ready\_queue = []

normal\_queue = []

temp = []

for i in range(len(process\_data)):

if(process\_data[i][1] <= s\_time and process\_data[i][3] == 0):

temp.extend([process\_data[i][0], process\_data[i][1],

process\_data[i][2], process\_data[i][4]])

ready\_queue.append(temp)

temp = []

elif(process\_data[i][3] == 0):

temp.extend([process\_data[i][0], process\_data[i][1],

process\_data[i][2], process\_data[i][4]])

19BQ5A0528 Page 15

normal\_queue.append(temp)

temp = []

if(len(ready\_queue) == 0 and len(normal\_queue) == 0):

break

if (len(ready\_queue) != 0):

ready\_queue.sort(key=lambda x: x[2])

start\_time.append(s\_time)

s\_time = s\_time + 1

e\_time = s\_time

exit\_time.append(e\_time)

sequence\_of\_process.append(ready\_queue[0][0])

for k in range(len(process\_data)):

if (process\_data[k][0] == ready\_queue[0][0]):

break

process\_data[k][2] = process\_data[k][2] - 1

if (process\_data[k][2] == 0): #If Burst Time of a process is 0, it

means the process is completed

process\_data[k][3] = 1

process\_data[k].append(e\_time)

if (len(ready\_queue) == 0):

if(s\_time < normal\_queue[0][1]):

s\_time = normal\_queue[0][1]

start\_time.append(s\_time)

s\_time = s\_time + 1

e\_time = s\_time

exit\_time.append(e\_time)

19BQ5A0528 Page 16

sequence\_of\_process.append(normal\_queue[0][0])

for k in range(len(process\_data)):

if (process\_data[k][0] == normal\_queue[0][0]):

break

process\_data[k][2] = process\_data[k][2] - 1

if (process\_data[k][2] == 0): #If Burst Time of a process is 0, it

means the process is completed

process\_data[k][3] = 1

process\_data[k].append(e\_time)

t\_time = Srtf.calculateTurnaroundTime(self, process\_data)

w\_time = Srtf.calculateWaitingTime(self, process\_data)

Srtf.printData(self, process\_data, t\_time, w\_time, sequence\_of\_process)

def calculateTurnaroundTime(self, process\_data):

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][5] - process\_data[i][1]

total\_turnaround\_time = total\_turnaround\_time + turnaround\_time

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time / len(process\_data)

return average\_turnaround\_time

def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

19BQ5A0528 Page 17

waiting\_time = process\_data[i][6] - process\_data[i][4]

total\_waiting\_time += waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time, sequence\_of\_process):

print("Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

if(j==2 or j==3):

continue

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

print('Average Waiting Time:',average\_waiting\_time)

context\_switch=0

for i in range(len(sequence\_of\_process)-1):

if(sequence\_of\_process[i]!=sequence\_of\_process[i+1]):

context\_switch+=1

print('Number of context switches:',context\_switch)

19BQ5A0528 Page 18



no\_of\_processes = int(input("Enter number of processes: "))

srtf = Srtf()

srtf.processData(no\_of\_processes)

**OUTPUT:**

Enter number of processes: 5

Enter Arrival Time for Process 1: 0

Enter Burst Time for Process 1: 7

Enter Arrival Time for Process 2: 2

Enter Burst Time for Process 2: 2

Enter Arrival Time for Process 3: 3

Enter Burst Time for Process 3: 5

Enter Arrival Time for Process 4: 5

Enter Burst Time for Process 4: 4

Enter Arrival Time for Process 5: 7

Enter Burst Time for Process 5: 6

Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time 1 0 7 9 9 2 2 2 2 4 2 0 3 3 5 18 15 10 4 5 4 13 8 4 5 7 6 24 17 11 Average Turnaround Time: 10.2

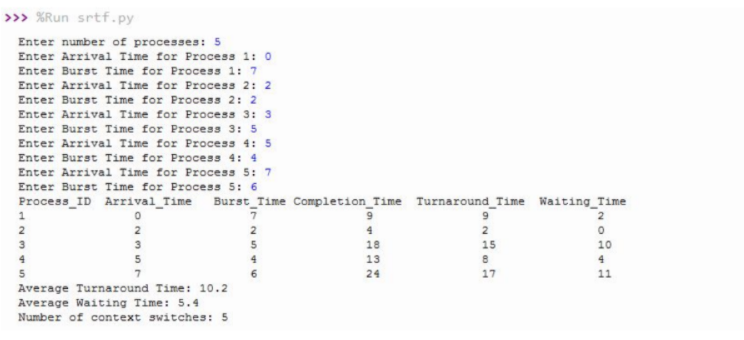
Average Waiting Time: 5.4

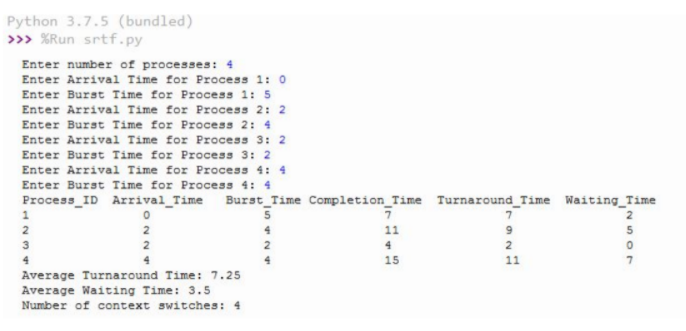
19BQ5A0528 Page 19

Number of context switches: 5

**OUTPUT SCREEN SHOTS:**

Output:1

Output:2

19BQ5A0528 Page 20

**EXPERIMENT NO: 1 (d)**

**AIM :** To implement program for Non-Preemptive Priority scheduling Algorithm

**DESCRIPTION :** The task is to find the Average Waiting Time and Average Turnaround Time of the given processes with their Burst Time using Priority Scheduling Algorithm.Priority is a scheduling policy that selects the waiting process with the highest priority to execute next.

Priority Scheduling is a Non Preemptive and Preemptive Algorithm, hence the process which has the Highest Priority is selected first.Here we are considering Non Preemptive version of Priority Scheduling, hence the process which has the Highest Priority will be served first and the next process will be served only after the previous process is executed completely.

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

Class priority:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class Priority:

def processData(self, no\_of\_processes):

19BQ5A0528 Page 21

process\_data = []

for i in range(1,no\_of\_processes+1):

temporary = []

arrival\_time = int(input("Enter Arrival Time for Process {}: ".format(i)))

burst\_time = int(input("Enter Burst Time for Process {}: ".format(i)))

priority = int(input("Enter Priority for Process {}: ".format(i)))

temporary.extend([i, arrival\_time, burst\_time, priority, 0])

process\_data.append(temporary)

Priority.schedulingProcess(self, process\_data)

def schedulingProcess(self, process\_data):

start\_time = []

exit\_time = []

s\_time = 0

process\_data.sort(key=lambda x: x[1])

for i in range(len(process\_data)):

ready\_queue = []

temp = []

19BQ5A0528 Page 22

normal\_queue = []

for j in range(len(process\_data)):

if((process\_data[j][1] <= s\_time) and (process\_data[j][4] == 0)):

temp.extend([process\_data[j][0], process\_data[j][1], process\_data[j][2],

process\_data[j][3]])

ready\_queue.append(temp)

temp = []

elif(process\_data[j][4] == 0):

temp.extend([process\_data[j][0], process\_data[j][1], process\_data[j][2],

process\_data[j][3]])

normal\_queue.append(temp)

temp = []

if(len(ready\_queue) != 0):

ready\_queue.sort(key=lambda x: x[3])

start\_time.append(s\_time)

s\_time = s\_time + ready\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

for k in range(len(process\_data)):

if(process\_data[k][0] == ready\_queue[0][0]):

break

process\_data[k][4] = 1

process\_data[k].append(e\_time)

elif (len(ready\_queue) == 0):

if (s\_time < normal\_queue[0][1]):

19BQ5A0528 Page 23

s\_time = normal\_queue[0][1]

start\_time.append(s\_time)

s\_time = s\_time + normal\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

for k in range(len(process\_data)):

if (process\_data[k][0] == normal\_queue[0][0]):

break

process\_data[k][4] = 1

process\_data[k].append(e\_time)

t\_time = Priority.calculateTurnaroundTime(self, process\_data)

w\_time = Priority.calculateWaitingTime(self, process\_data)

Priority.printData(self, process\_data, t\_time, w\_time)

def calculateTurnaroundTime(self, process\_data):

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][5] - process\_data[i][1]

total\_turnaround\_time = total\_turnaround\_time + turnaround\_time

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time / len(process\_data)

return average\_turnaround\_time

19BQ5A0528 Page 24



def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

waiting\_time = process\_data[i][6] - process\_data[i][2]

total\_waiting\_time = total\_waiting\_time + waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time):

print("Process\_ID Arrival\_Time Burst\_Time Priority Completion\_Time Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

if(j==4):

continue

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

19BQ5A0528 Page 25

print('Average Waiting Time:',average\_waiting\_time)

print('Number of context switches:',no\_of\_processes-1)

no\_of\_processes = int(input("Enter number of processes: "))

priority = Priority()

priority.processData(no\_of\_processes)

**OUTPUT:**

Enter number of processes: 5

Enter Arrival Time for Process 1: 0

Enter Burst Time for Process 1: 7

Enter Priority for Process 1: 3

Enter Arrival Time for Process 2: 2

Enter Burst Time for Process 2: 2

Enter Priority for Process 2: 2

Enter Arrival Time for Process 3: 3

Enter Burst Time for Process 3: 5

Enter Priority for Process 3: 1

Enter Arrival Time for Process 4: 5

Enter Burst Time for Process 4: 4

Enter Priority for Process 4: 4

Enter Arrival Time for Process 5: 7

Enter Burst Time for Process 5: 6

Enter Priority for Process 5: 6

19BQ5A0528 Page 26

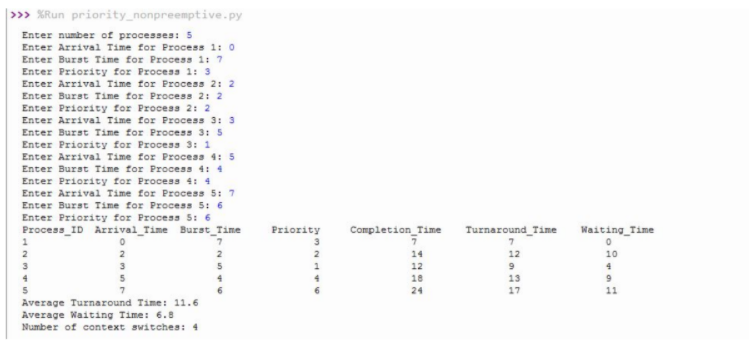
Process\_ID Arrival\_Time Burst\_Time Priority Completion\_Time Turnaround\_Time Waiting\_Time 1 0 7 3 7 7 0 2 2 2 2 14 12 10 3 3 5 1 12 9 4 4 5 4 4 18 13 9 5 7 6 6 24 17 11

Average Turnaround Time: 11.6

Average Waiting Time: 6.8

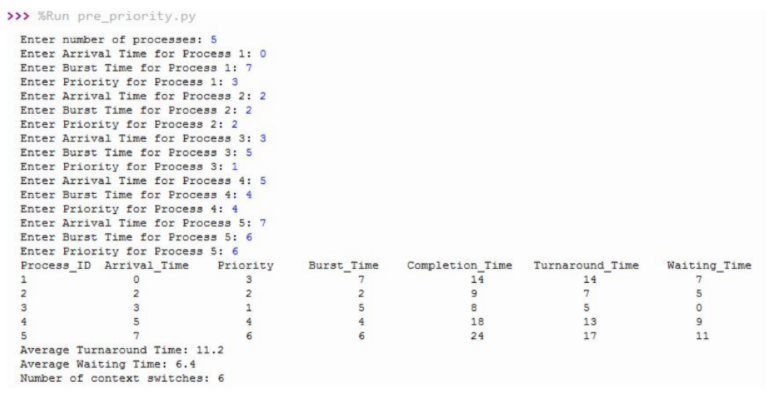
Number of context switches: 4

**OUTPUT SCREENSHOTS:**

****19BQ5A0528 Page 27



**Output:2**

****

19BQ5A0528 Page 28

**EXPERIMENT NO: 1 (e)**

**AIM :** To implement program for Preemptive Priority scheduling Algorithm

**DESCRIPTION :** The task is to find the Average Waiting Time and Average Turnaround Time of the given processes with their Burst Time using Priority Scheduling Algorithm.

Priority is a scheduling policy that selects the waiting process with the highest priority to execute next.Priority Scheduling is a Non Preemptive and Preemptive Algorithm, hence the process which has the Highest Priority is selected first.

Here we are considering Preemptive version of Priority Scheduling, hence the process which has the Highest Priority will be served first and will be continued to be served till there is any other process with higher priority.If there is any process with higher priority, then switch the process.

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

Class Pre-priority:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class Pre\_Priority:

def processData(self, no\_of\_processes):

process\_data = []

19BQ5A0528 Page 29

for i in range(1,no\_of\_processes+1):

temporary = []

arrival\_time = int(input("Enter Arrival Time for Process {}: ".format(i)))

burst\_time = int(input("Enter Burst Time for Process {}: ".format(i)))

priority = int(input("Enter Priority for Process {}: ".format(i)))

temporary.extend([i, arrival\_time, burst\_time, priority, 0, burst\_time])

process\_data.append(temporary)

Priority.schedulingProcess(self, process\_data)

def schedulingProcess(self, process\_data):

start\_time = []

exit\_time = []

s\_time = 0

sequence\_of\_process = []

process\_data.sort(key=lambda x: x[1])

while (1):

ready\_queue = []

normal\_queue = []

19BQ5A0528Page 30

temp = []

for i in range(len(process\_data)):

if (process\_data[i][1] <= s\_time and process\_data[i][4] == 0):

temp.extend([process\_data[i][0], process\_data[i][1], process\_data[i][2],

process\_data[i][3],

process\_data[i][5]])

ready\_queue.append(temp)

temp = []

elif (process\_data[i][4] == 0):

temp.extend([process\_data[i][0], process\_data[i][1], process\_data[i][2],

process\_data[i][4],

process\_data[i][5]])

normal\_queue.append(temp)

temp = []

if (len(ready\_queue) == 0 and len(normal\_queue) == 0):

break

if (len(ready\_queue) != 0):

ready\_queue.sort(key=lambda x: x[3])

start\_time.append(s\_time)

s\_time = s\_time + 1

e\_time = s\_time

exit\_time.append(e\_time)

sequence\_of\_process.append(ready\_queue[0][0])

for k in range(len(process\_data)):

if process\_data[k][0] == ready\_queue[0][0]:

break

19BQ5A0528Page 31

process\_data[k][2] = process\_data[k][2] - 1

if (process\_data[k][2] == 0): #if burst time is zero, it means process is completed process\_data[k][4] = 1

process\_data[k].append(e\_time)

if (len(ready\_queue) == 0):

normal\_queue.sort(key=lambda x: x[1])

if (s\_time < normal\_queue[0][1]):

s\_time = normal\_queue[0][1]

start\_time.append(s\_time)

s\_time = s\_time + 1

e\_time = s\_time

exit\_time.append(e\_time)

sequence\_of\_process.append(normal\_queue[0][0])

for k in range(len(process\_data)):

if (process\_data[k][0] == normal\_queue[0][0]):

break

process\_data[k][2] = process\_data[k][2] - 1

if (process\_data[k][2] == 0): #if burst time is zero, it means process is completed process\_data[k][4] = 1

process\_data[k].append(e\_time)

t\_time = Priority.calculateTurnaroundTime(self, process\_data)

w\_time = Priority.calculateWaitingTime(self, process\_data)

Priority.printData(self, process\_data, t\_time, w\_time, sequence\_of\_process)

def calculateTurnaroundTime(self, process\_data):

19BQ5A0528 Page 32

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][6] - process\_data[i][1]

total\_turnaround\_time = total\_turnaround\_time + turnaround\_time

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time / len(process\_data)

return average\_turnaround\_time

def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

waiting\_time = process\_data[i][7] - process\_data[i][5]

total\_waiting\_time = total\_waiting\_time + waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time, sequence\_of\_process):

print("Process\_ID Arrival\_Time Priority Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

if(j==2 or j==4):

19BQ5A0528 Page 33

continue

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

print('Average Waiting Time:',average\_waiting\_time)

context\_switch=0

for i in range(len(sequence\_of\_process)-1):

if(sequence\_of\_process[i]!=sequence\_of\_process[i+1]):

context\_switch+=1

print('Number of context switches:',context\_switch)

no\_of\_processes = int(input("Enter number of processes: "))

priority = Priority()

priority.processData(no\_of\_processes)

**Output:**

Enter number of processes: 5

Enter Arrival Time for Process 1: 0

19BQ5A0528 Page 34



Enter Burst Time for Process 1: 7

Enter Priority for Process 1: 3

Enter Arrival Time for Process 2: 2

Enter Burst Time for Process 2: 2

Enter Priority for Process 2: 2

Enter Arrival Time for Process 3: 3

Enter Burst Time for Process 3: 5

Enter Priority for Process 3: 1

Enter Arrival Time for Process 4: 5

Enter Burst Time for Process 4: 4

Enter Priority for Process 4: 4

Enter Arrival Time for Process 5: 7

Enter Burst Time for Process 5: 6

Enter Priority for Process 5: 6

Process\_ID Arrival\_Time Priority Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time 1 0 3 7 14 14 7 2 2 2 2 9 7 5 3 3 1 5 8 5 0 4 5 4 4 18 13 9 5 7 6 6 24 17 11 Average Turnaround Time: 11.2

Average Waiting Time: 6.4

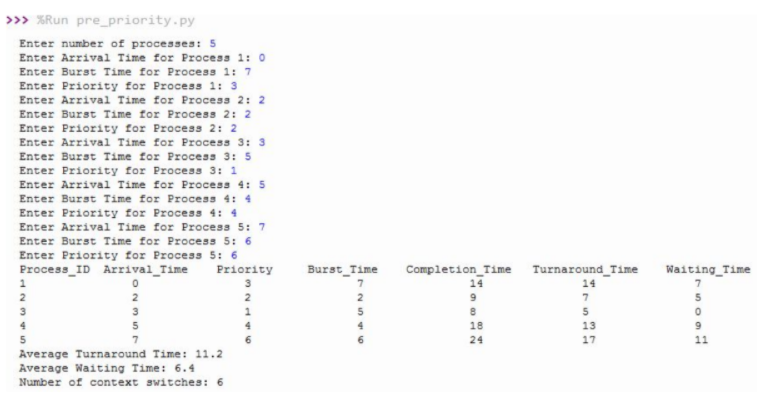
Number of context switches: 6

**OUTPUT SCREENSHOTS:**

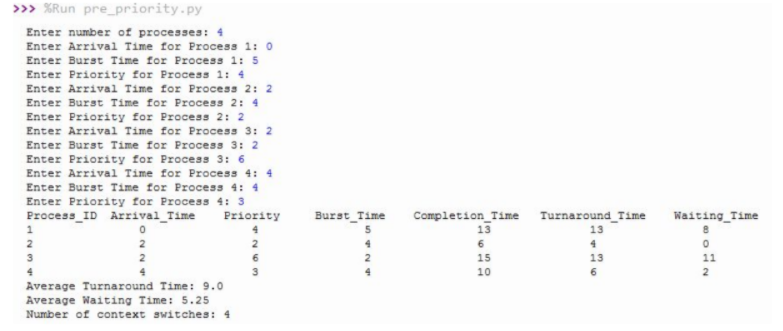
**Output:1**

19BQ5A0528 Page 35





**Output:2**

****

19BQ5A0528 Page 36

**EXPERIMENT NO: 1 (f)**

**AIM :** To implement program for Round Robin scheduling Algorithm

**DESCRIPTION :** The task is to find the Average Waiting Time and Average Turnaround Time of the given processes with their Burst Time using Round Robin Scheduling Algorithm.

Round Robin is a scheduling policy that selects the waiting process and executes it for a fixed time quantumRound Robin is a Preemptive Algorithm, hence the process will execute for a fixed time quantum and then it is switched and another process is executedRound Robin is cyclic in nature so it does not cause starvation.We will consider processes having different Arrival Time

**PROGRAMMING LANGUAGE USED:** PYTHON

**LIBRARIES USED:** No built-in libraries used

**SYNTAX:**

Class Round robin:

processData(self,no\_of\_processes)

schedulingProcess(self,process\_data)

calculateTurnaroundTime(self,process\_data)

calculateWaitingTime(self,process\_data)

printData(self,process\_data,average\_turnaround\_time,average\_waiting\_time)

**PROGRAM:**

class RoundRobin:

def processData(self, no\_of\_processes):

process\_data = []

for i in range(1,no\_of\_processes+1):

temporary = []

19BQ5A0528 Page 37



arrival\_time = int(input("Enter Arrival Time for Process {}: ".format(i)))

burst\_time = int(input("Enter Burst Time for Process {}: ".format(i)))

temporary.extend([i, arrival\_time, burst\_time, 0, burst\_time])

process\_data.append(temporary)

time\_slice = int(input("Enter Time Slice: "))

RoundRobin.schedulingProcess(self, process\_data, time\_slice)

def schedulingProcess(self, process\_data, time\_slice):

start\_time = []

exit\_time = []

executed\_process = []

ready\_queue = []

s\_time = 0

while 1:

normal\_queue = []

temp = []

for i in range(len(process\_data)):

if( process\_data[i][1] <= s\_time and process\_data[i][3] == 0):

19BQ5A0528 Page 38



present = 0

if (len(ready\_queue) != 0):

for k in range(len(ready\_queue)):

if (process\_data[i][0] == ready\_queue[k][0]):

present = 1

if(present == 0):

temp.extend([process\_data[i][0], process\_data[i][1], process\_data[i][2],

process\_data[i][4]])

ready\_queue.append(temp)

temp = []

if (len(ready\_queue) != 0 and len(executed\_process) != 0):

for k in range(len(ready\_queue)):

if (ready\_queue[k][0] == executed\_process[len(executed\_process) - 1]):

ready\_queue.insert((len(ready\_queue) - 1), ready\_queue.pop(k))

elif (process\_data[i][3] == 0):

temp.extend([process\_data[i][0], process\_data[i][1], process\_data[i][2],

process\_data[i][4]])

normal\_queue.append(temp)

temp = []

if (len(ready\_queue) == 0 and len(normal\_queue) == 0):

break

if (len(ready\_queue) != 0):

if (ready\_queue[0][2] > time\_slice):

19BQ5A0528 Page 39



start\_time.append(s\_time)

s\_time = s\_time + time\_slice

e\_time = s\_time

exit\_time.append(e\_time)

executed\_process.append(ready\_queue[0][0])

for j in range(len(process\_data)):

if(process\_data[j][0] == ready\_queue[0][0]):

break

process\_data[j][2] = process\_data[j][2] - time\_slice

ready\_queue.pop(0)

elif (ready\_queue[0][2] <= time\_slice):

start\_time.append(s\_time)

s\_time = s\_time + ready\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

executed\_process.append(ready\_queue[0][0])

for j in range(len(process\_data)):

if( process\_data[j][0] == ready\_queue[0][0]):

break

process\_data[j][2] = 0

process\_data[j][3] = 1

process\_data[j].append(e\_time)

ready\_queue.pop(0)

19BQ5A0528 Page 40



elif (len(ready\_queue) == 0):

if (s\_time < normal\_queue[0][1]):

s\_time = normal\_queue[0][1]

if (normal\_queue[0][2] > time\_slice):

start\_time.append(s\_time)

s\_time = s\_time + time\_slice

e\_time = s\_time

exit\_time.append(e\_time)

executed\_process.append(normal\_queue[0][0])

for j in range(len(process\_data)):

if process\_data[j][0] == normal\_queue[0][0]:

break

process\_data[j][2] = process\_data[j][2] - time\_slice

elif (normal\_queue[0][2] <= time\_slice):

start\_time.append(s\_time)

s\_time = s\_time + normal\_queue[0][2]

e\_time = s\_time

exit\_time.append(e\_time)

executed\_process.append(normal\_queue[0][0])

for j in range(len(process\_data)):

if (process\_data[j][0] == normal\_queue[0][0]):

break

process\_data[j][2] = 0

19BQ5A0528 Page 41



process\_data[j][3] = 1

process\_data[j].append(e\_time)

t\_time = RoundRobin.calculateTurnaroundTime(self, process\_data)

w\_time = RoundRobin.calculateWaitingTime(self, process\_data)

RoundRobin.printData(self, process\_data, t\_time, w\_time, executed\_process)

def calculateTurnaroundTime(self, process\_data):

total\_turnaround\_time = 0

for i in range(len(process\_data)):

turnaround\_time = process\_data[i][5] - process\_data[i][1]

total\_turnaround\_time = total\_turnaround\_time + turnaround\_time

process\_data[i].append(turnaround\_time)

average\_turnaround\_time = total\_turnaround\_time / len(process\_data)

return average\_turnaround\_time

def calculateWaitingTime(self, process\_data):

total\_waiting\_time = 0

for i in range(len(process\_data)):

waiting\_time = process\_data[i][6] - process\_data[i][4]

#waiting\_time = turnaround\_time - burst\_time

total\_waiting\_time = total\_waiting\_time + waiting\_time

process\_data[i].append(waiting\_time)

average\_waiting\_time = total\_waiting\_time / len(process\_data)

19BQ5A0528 Page 42



#average\_waiting\_time = total\_waiting\_time / no\_of\_processes

return average\_waiting\_time

def printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time, executed\_process):

process\_data.sort(key=lambda x: x[0])

print("Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time")

for i in range(len(process\_data)):

for j in range(len(process\_data[i])):

if(j==2 or j==3):

continue

print(process\_data[i][j], end="\t\t")

print()

print('Average Turnaround Time:',average\_turnaround\_time)

print('Average Waiting Time:',average\_waiting\_time)

context\_switch=0

for i in range(len(executed\_process)-1):

if(executed\_process[i]!=executed\_process[i+1]):

context\_switch+=1

print('Number of context switches:',context\_switch )

19BQ5A0528 Page 43



no\_of\_processes = int(input("Enter number of processes: "))

rr = RoundRobin()

rr.processData(no\_of\_processes)

**OUTPUT:**

Enter number of processes: 5

Enter Arrival Time for Process 1: 0

Enter Burst Time for Process 1: 7

Enter Arrival Time for Process 2: 2

Enter Burst Time for Process 2: 2

Enter Arrival Time for Process 3: 3

Enter Burst Time for Process 3: 5

Enter Arrival Time for Process 4: 5

Enter Burst Time for Process 4: 4

Enter Arrival Time for Process 5: 7

Enter Burst Time for Process 5: 6

Enter Time Slice: 3

Process\_ID Arrival\_Time Burst\_Time Completion\_Time Turnaround\_Time Waiting\_Time 1 0 7 20 20 13 2 2 2 5 3 1 3 3 5 19 16 11 4 5 4 21 16 12 5 7 6 24 17 11 Average Turnaround Time: 14.4

19BQ5A0528 Page 44

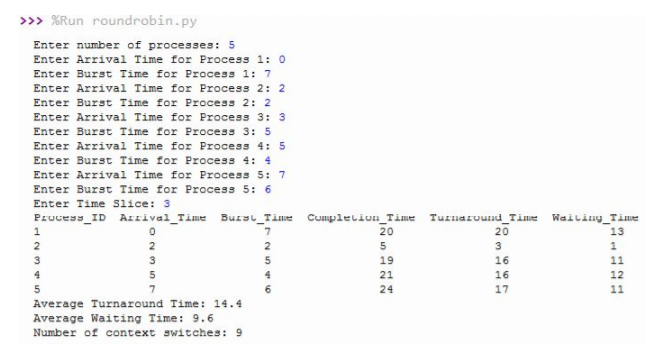


Average Waiting Time: 9.6

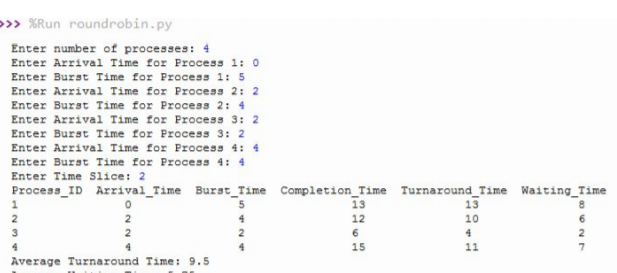
Number of context switches: 9

**OUTPUT SCREENSHOTS:**

**Output:1**

****

**Output:2**

****

19BQ5A0528 Page 45