Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Lab Report #2

[Course title: COMP 307]

Submitted by:
Bisheshwor Neupane(35)

Submitted to:
Mr. Dhiraj Shrestha
Department of Computer Science and Engineering

Submission Date:

August 14, 2021

1. **Consider the Reader writer problem and provide the solutions for Race condition using peterson solutions**

Solution:

Assume that the LOAD and STORE instructions are atomic; i.e. cannot be interrupted.

The two processes share two variables:

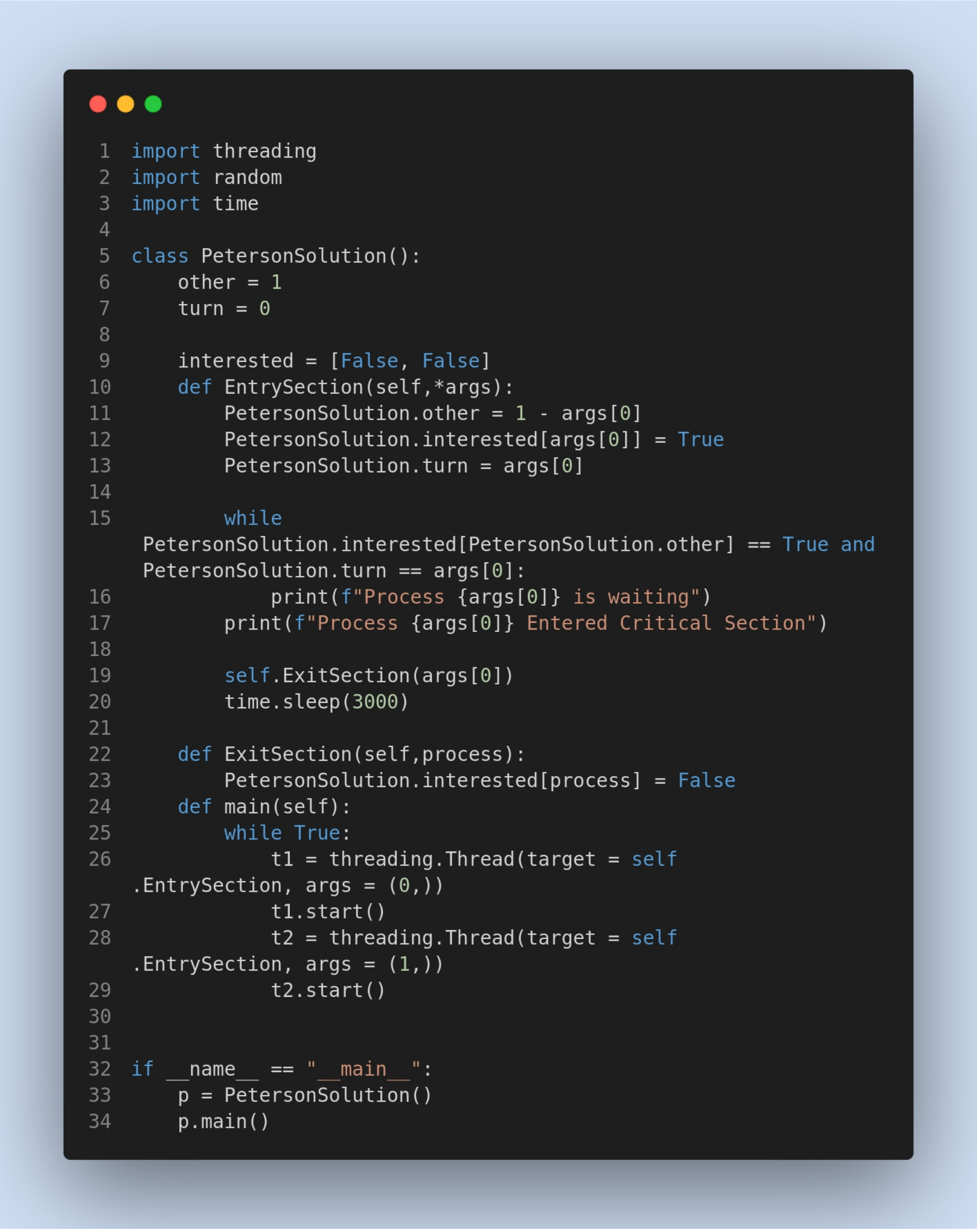
int turn;
Boolean flag[2]

The variable turn indicated whose turn it is to enter the critical section.

The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = true implies that process Pi is ready.

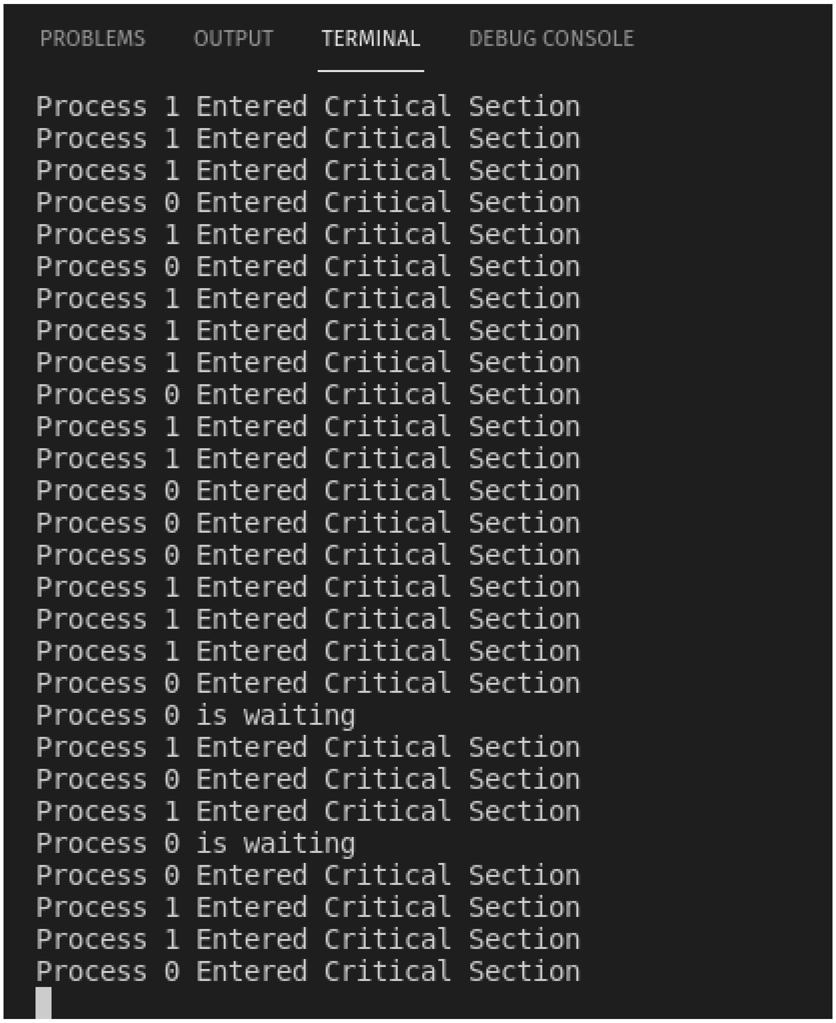
```
Algorithm for Process Pi:
do{
        flag[i] = TRUE;
        turn = j;
        while(flag[j] && turn == j);

        critical section

        flag[i] = FALSE;
                remainder section
}while(TRUE);

Solution to Critical-section problem using locks
do{
        aquire lock
                critical section
        release lock
                remainder section
}while(TRUE);
```

Implementing it in Python:



```python
import threading
import random
import time

class PetersonSolution():
    other = 1
    turn = 0

    interested = [False, False]
    def EntrySection(self,*args):
        PetersonSolution.other = 1 - args[0]
        PetersonSolution.interested[args[0]] = True
        PetersonSolution.turn = args[0]

        while PetersonSolution.interested[PetersonSolution.other] == True and PetersonSolution.turn == args[0]:
            print(f"Process {args[0]} is waiting")
        print(f"Process {args[0]} Entered Critical Section")

        self.ExitSection(args[0])
        time.sleep(3000)

    def ExitSection(self,process):
        PetersonSolution.interested[process] = False
    def main(self):
        while True:
            t1 = threading.Thread(target = self.EntrySection, args = (0,))
            t1.start()
            t2 = threading.Thread(target = self.EntrySection, args = (1,))
            t2.start()


if __name__ == "__main__":
    p = PetersonSolution()
    p.main()
```

Output:



```
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 0 Entered Critical Section
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 0 is waiting
Process 1 Entered Critical Section
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 0 is waiting
Process 0 Entered Critical Section
Process 1 Entered Critical Section
Process 1 Entered Critical Section
Process 0 Entered Critical Section
```

2. Calculate Waiting time, Turnaround time for each process and also calculate the average Turnaround time and Average Waiting time (Assume 5 different process with different arrival time and burst time):

a. First Come First Serve

b. Round Robin

c. Shortest Job First

d. Priority

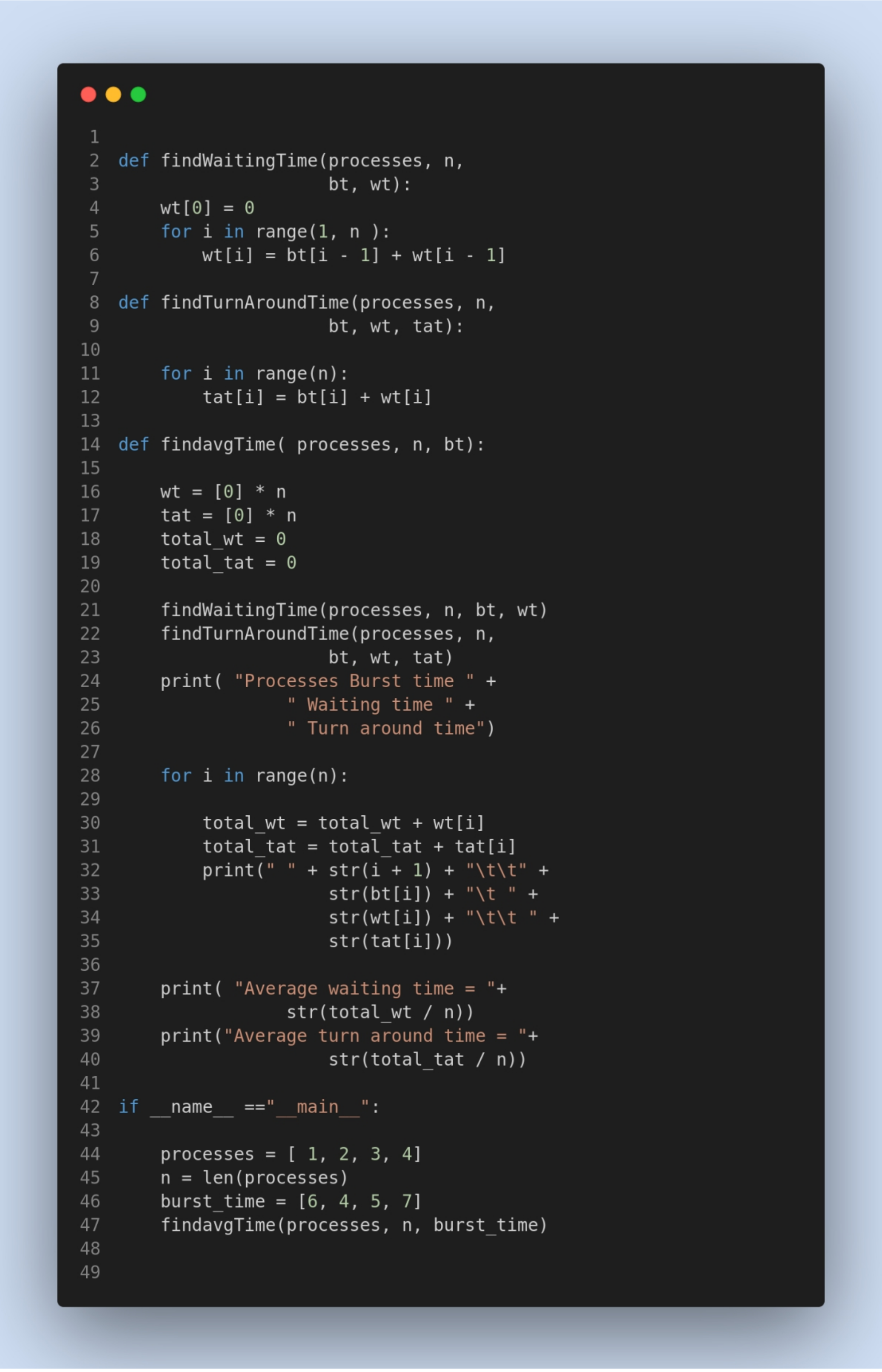
Solution:

i. First Come First Serve

Algorithm:

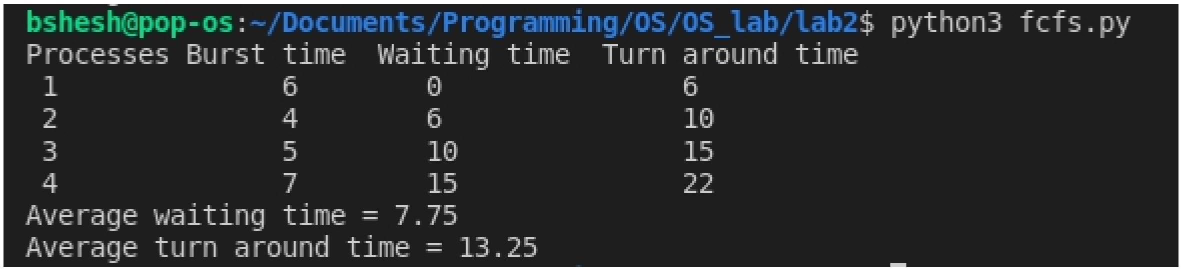
```
1- Input the processes along with their burst time (bt).
2- Find waiting time (wt) for all processes.
3- As first process that comes need not to wait so
   waiting time for process 1 will be 0 i.e. wt[0] = 0.
4- Find waiting time for all other processes i.e. for
    process i ->
        wt[i] = bt[i-1] + wt[i-1] .
5- Find turnaround time = waiting_time + burst_time
   for all processes.
6- Find average waiting time =
            total_waiting_time / no_of_processes.
7- Similarly, find average turnaround time =
            total_turn_around_time / no_of_processes.
```

Implementation in Python:



```python

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))

if __name__ =="__main__":

    processes = [ 1, 2, 3, 4]
    n = len(processes)
    burst_time = [6, 4, 5, 7]
    findavgTime(processes, n, burst_time)


```

Output:



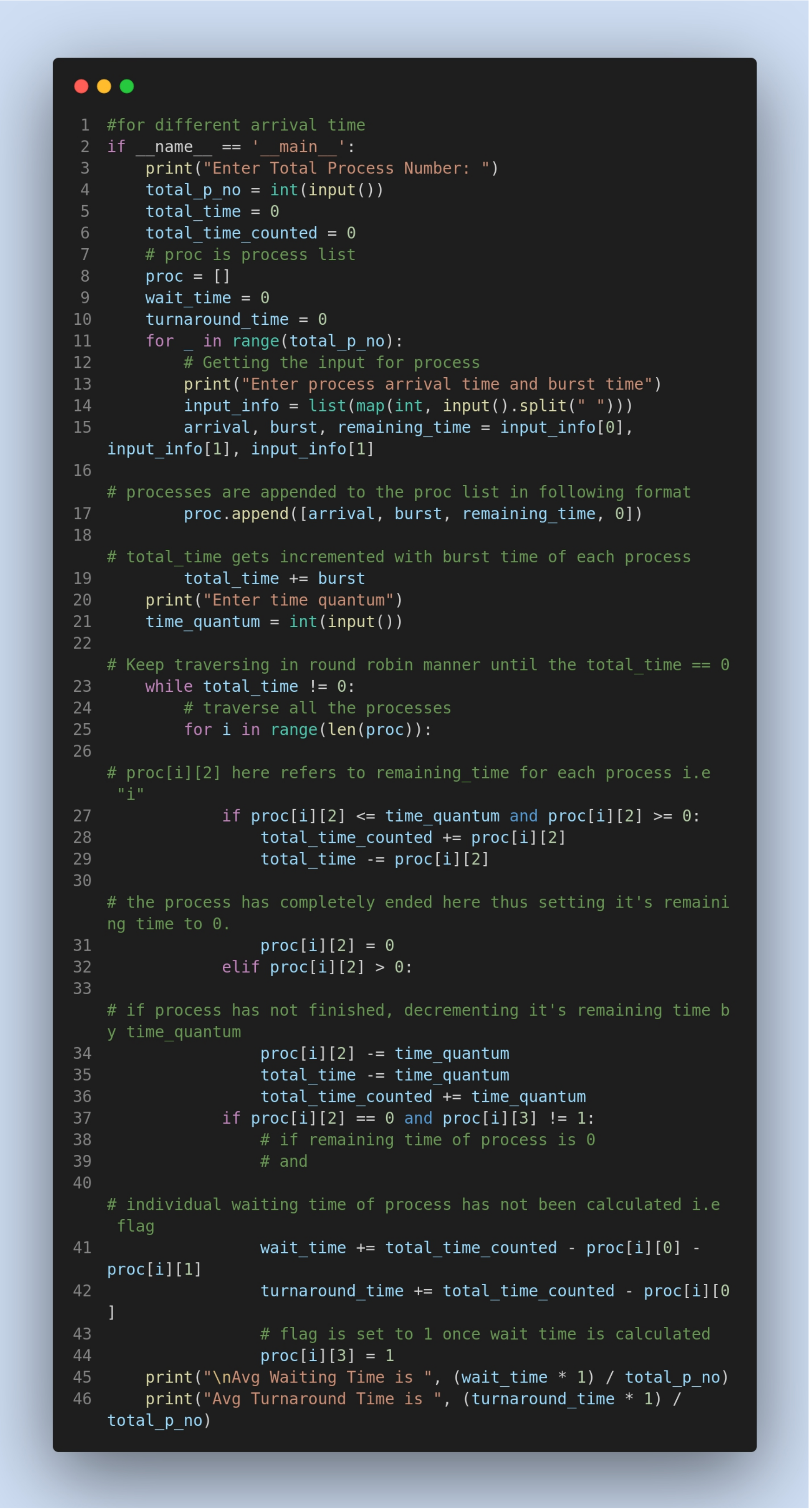
```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab2$ python3 fcfs.py
Processes Burst time  Waiting time  Turn around time
 1              6       0             6
 2              4       6             10
 3              5       10            15
 4              7       15            22
Average waiting time = 7.75
Average turn around time = 13.25
```

ii. Round Robin:

Algorithm:

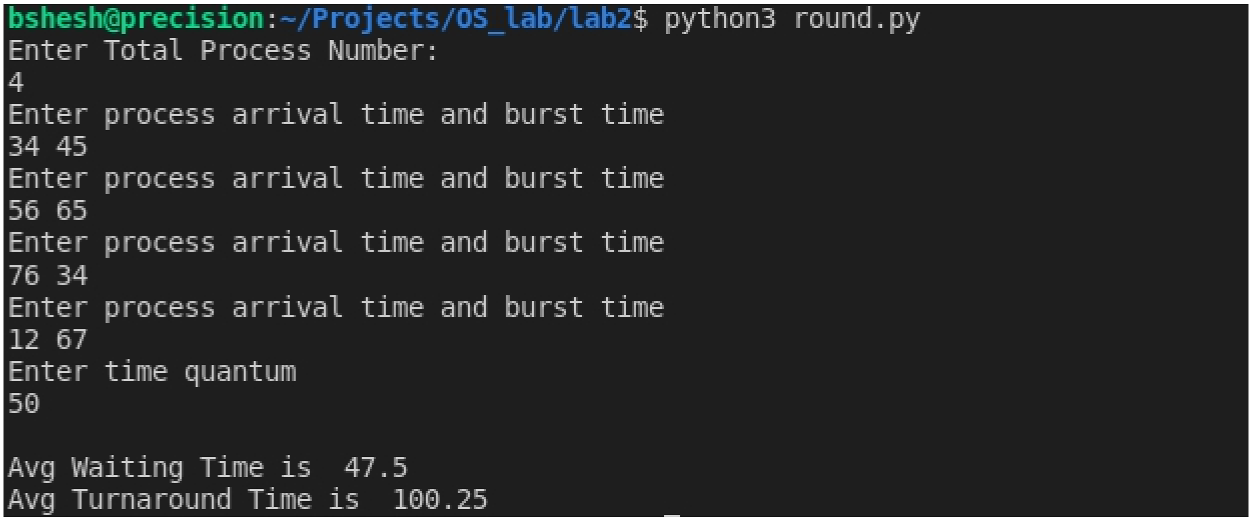
```
1- Create an array rem_bt[] to keep track of remaining
   burst time of processes. This array is initially a
   copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times
   of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes
   are not done. Do following for i'th process if it is
   not done yet.
    a- If rem_bt[i] > quantum
       (i)  t = t + quantum
       (ii) bt_rem[i] -= quantum;
    c- Else // Last cycle for this process
       (i)  t = t + bt_rem[i];
       (ii) wt[i] = t - bt[i]
       (ii) bt_rem[i] = 0; // This process is over
```

Implementation in Python:



```python
#for different arrival time
if __name__ == '__main__':
    print("Enter Total Process Number: ")
    total_p_no = int(input())
    total_time = 0
    total_time_counted = 0
    # proc is process list
    proc = []
    wait_time = 0
    turnaround_time = 0
    for _ in range(total_p_no):
        # Getting the input for process
        print("Enter process arrival time and burst time")
        input_info = list(map(int, input().split(" ")))
        arrival, burst, remaining_time = input_info[0], input_info[1], input_info[1]

# processes are appended to the proc list in following format
        proc.append([arrival, burst, remaining_time, 0])

# total_time gets incremented with burst time of each process
        total_time += burst
    print("Enter time quantum")
    time_quantum = int(input())

# Keep traversing in round robin manner until the total_time == 0
    while total_time != 0:
        # traverse all the processes
        for i in range(len(proc)):

# proc[i][2] here refers to remaining_time for each process i.e "i"
            if proc[i][2] <= time_quantum and proc[i][2] >= 0:
                total_time_counted += proc[i][2]
                total_time -= proc[i][2]

# the process has completely ended here thus setting it's remaining time to 0.
                proc[i][2] = 0
            elif proc[i][2] > 0:

# if process has not finished, decrementing it's remaining time by time_quantum
                proc[i][2] -= time_quantum
                total_time -= time_quantum
                total_time_counted += time_quantum
            if proc[i][2] == 0 and proc[i][3] != 1:
                # if remaining time of process is 0
                # and

# individual waiting time of process has not been calculated i.e flag
                wait_time += total_time_counted - proc[i][0] - proc[i][1]
                turnaround_time += total_time_counted - proc[i][0]
                # flag is set to 1 once wait time is calculated
                proc[i][3] = 1
    print("\nAvg Waiting Time is ", (wait_time * 1) / total_p_no)
    print("Avg Turnaround Time is ", (turnaround_time * 1) / total_p_no)
```

Output:



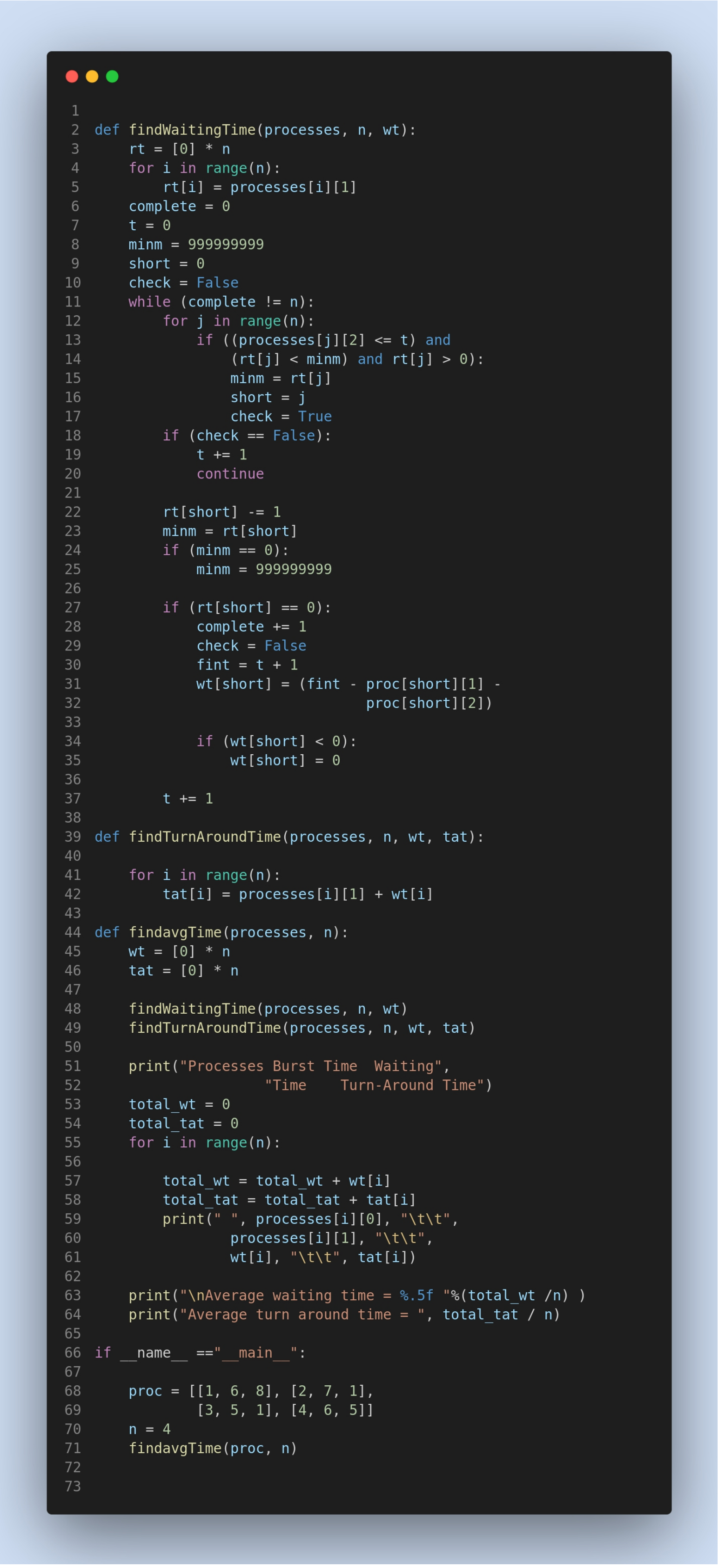
```
bshesh@precision:~/Projects/OS_lab/lab2$ python3 round.py
Enter Total Process Number:
4
Enter process arrival time and burst time
34 45
Enter process arrival time and burst time
56 65
Enter process arrival time and burst time
76 34
Enter process arrival time and burst time
12 67
Enter time quantum
50

Avg Waiting Time is  47.5
Avg Turnaround Time is  100.25
```

iii. Shortest Job First:

Algorithm:

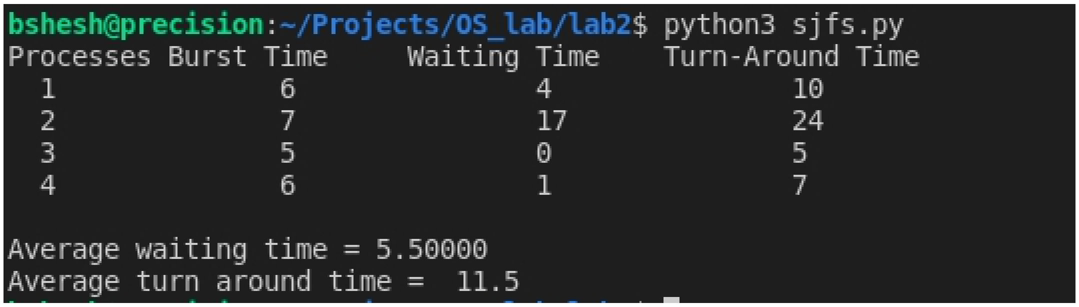
1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

Implementation in Python:



```python

def findWaitingTime(processes, n, wt):
    rt = [0] * n
    for i in range(n):
        rt[i] = processes[i][1]
    complete = 0
    t = 0
    minm = 999999999
    short = 0
    check = False
    while (complete != n):
        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

        rt[short] -= 1
        minm = rt[short]
        if (minm == 0):
            minm = 999999999

        if (rt[short] == 0):
            complete += 1
            check = False
            fint = t + 1
            wt[short] = (fint - proc[short][1] -
                                proc[short][2])

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

        t += 1

def findTurnAroundTime(processes, n, wt, tat):

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

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

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

    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)

if __name__ =="__main__":

    proc = [[1, 6, 8], [2, 7, 1],
            [3, 5, 1], [4, 6, 5]]
    n = 4
    findavgTime(proc, n)


```

Output:



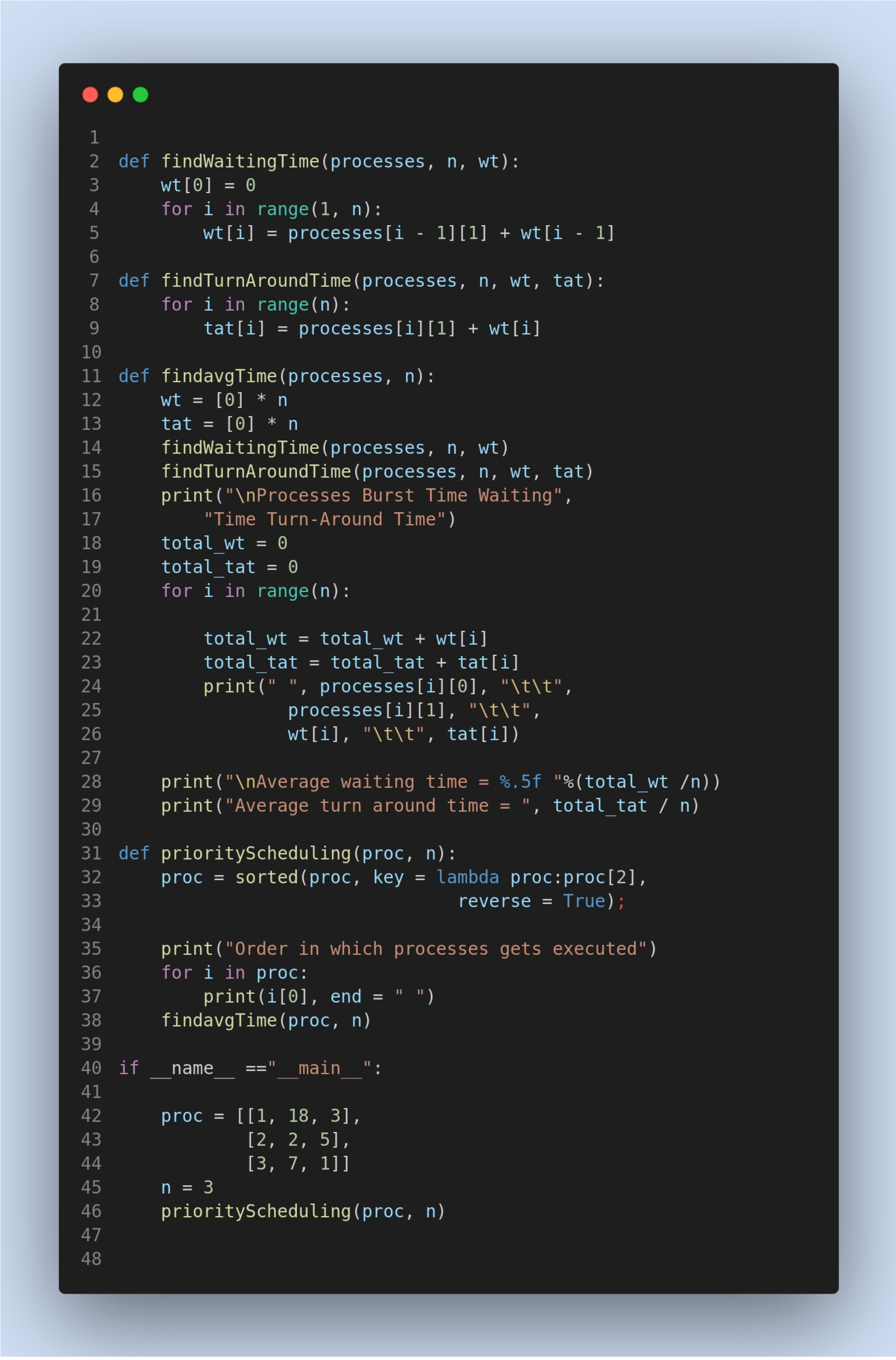
```
bshesh@precision:~/Projects/OS_lab/lab2$ python3 sjfs.py
Processes Burst Time     Waiting Time    Turn-Around Time
  1             6               4               10
  2             7               17              24
  3             5               0               5
  4             6               1               7

Average waiting time = 5.50000
Average turn around time =  11.5
```

iv. Priority Scheduling:

Algorithm:

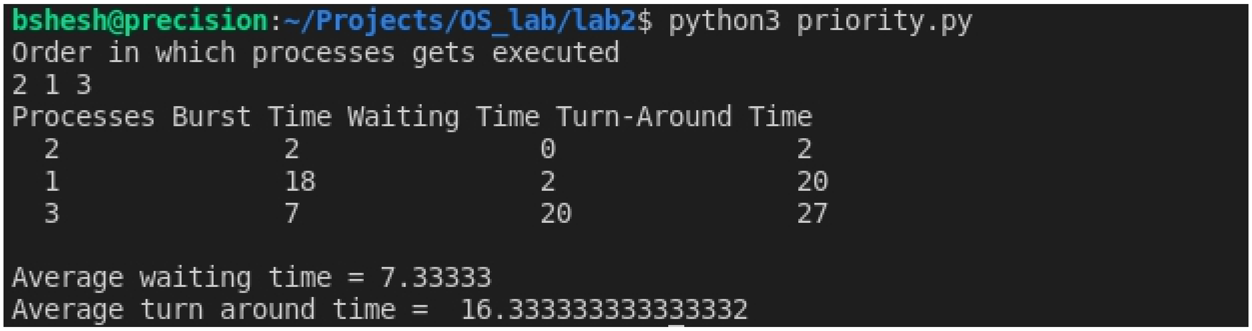
1. First input the processes with their arrival time, burst time and priority.
2. First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
3. Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number. Note: In the question, They will clearly mention, which number will have higher priority and which number will have lower priority.
4. Once all the processes have been arrived, we can schedule them based on their priority.

Implementation in Python:



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

def findTurnAroundTime(processes, n, wt, tat):
    for i in range(n):
        tat[i] = processes[i][1] + wt[i]

def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n
    findWaitingTime(processes, n, wt)
    findTurnAroundTime(processes, n, wt, tat)
    print("\nProcesses 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)

def priorityScheduling(proc, n):
    proc = sorted(proc, key = lambda proc:proc[2],
                                reverse = True);

    print("Order in which processes gets executed")
    for i in proc:
        print(i[0], end = " ")
    findavgTime(proc, n)

if __name__ =="__main__":

    proc = [[1, 18, 3],
            [2, 2, 5],
            [3, 7, 1]]
    n = 3
    priorityScheduling(proc, n)
```

Output:



```
bshesh@precision:~/Projects/OS_lab/lab2$ python3 priority.py
Order in which processes gets executed
2 1 3
Processes Burst Time Waiting Time Turn-Around Time
  2              2               0               2
  1              18              2               20
  3              7               20              27

Average waiting time = 7.33333
Average turn around time =  16.333333333333332
```