**Student Name: Roopak Parashar**

**Student ID:** 11808599 (A05)

**Email Address:** [roopakparashar17@gmail.com](mailto:devesh.chaudhary.30994@gmail.com)

**GitHub Link:**

Problem:

**Ques.**

CPU schedules N processes which arrive at different time intervals and each process is allocated the a specific user input time unit, processes are scheduled using a preemptive round robin scheduling algorithm. Each process must be assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes one task has priority 0. The length of a time quantum is T units, where T is the custom time considered as time quantum for processing. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue. Design a scheduler so that the task with priority 0 does not starve for resources and gets the CPU at some time unit to execute. Also compute waiting time, turn around.

**Ans:** We will now solve the problem by preemptive round robin Scheduling:

## **What is Round-Robin Scheduling?**

## The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking.In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

**Algorithm:**

* Start the process
* CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
* This fixed amount of time is called as **time quantum** or **time slice**.
* After the time quantum expires, the running process is preempted and sent to the ready queue.
* Then, the processor is assigned to the next arrived process.
* It is always preemptive in nature.
* **average waiting** and **Turn around time**.

1. Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1 )+ the time difference in getting the CPU from process(n-1)
2. Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

* **Caculate:**

1. Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process

* Stop the process.

Code Snippet:-

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

float avg\_wait\_time(int wt[], int n)

{

float x = 0;

int i,sum = 0;

for(i=0;i<n;i++)

sum = sum + wt[i];

x = sum \* 1.0;

x = x / n;

return x;

}

float avg\_turnaround\_time(int tat[], int n)

{

float x = 0;

int i,sum = 0;

for(i=0;i<n;i++)

sum = sum + tat[i];

x = sum \* 1.0;

x = x / n;

return x;

}

void rearrange\_process\_queue(int pq[],int rt[],int pty[],int n,int running\_processes)

{

int i;

if(pty[0]<pty[1])

{

int temp = pq[0];

for(i=0;i<running\_processes;i++)

{

pq[i] = pq[i+1];

}

pq[running\_processes-1] = temp;

}

if(rt[pq[0]-1]==0)

{

int temp = pq[0];

for(i=0;i<running\_processes;i++)

{

pq[i] = pq[i+1];

}

pq[running\_processes-1] = temp;

running\_processes=running\_processes-1;

}

}

void minptyinc(int pty[],int n)

{

int i,min=pty[0];

for(i=1;i<n;i++)

{

if(min>=pty[i])

min=pty[i];

}

for(i=0;i<n;i++)

{

if(pty[i]==min)

pty[i]++;

}

}

int main()

{

int count,i,j;

int time\_quantum,n;

int time = 0;

printf("\nEnter Number of Processes =");

scanf("%d" ,&n);

printf("\nEnter Time Quantum =");

scanf("%d" ,&time\_quantum);

if(n <= 0 || time\_quantum <= 0)

{

printf("Invalid data!");

return 0;

}

printf("The number of processes are set to: %d\nThe time quantum is set to:%d\n" , n,time\_quantum);

int at[10],bt[10],rt[10],pq[10],pty[10],pty1[10],pflag[10],tat[10],wt[10];

for(j=0;j<n;j++)

{

pq[j] = 0;

pflag[j] = 0;

}

for(count=0;count<n;count++)

{

printf("\nEnter Detail for Process = %d",count+1);

printf("\nEnter Arrival time = ");

scanf("%d",&at[count]);

printf("Enter Burst time = ");

scanf("%d",&bt[count]);

printf("Enter Priority = ");

scanf("%d",&pty[count]);

rt[count]=bt[count];

pty1[count]=pty[count];

if(at[count] < 0 || bt[count] <= 0)

{

printf("Invalid Data!");

return 0;

}

}

int current = 0;

int running\_processes = 0;

int x=0;

pq[0] = 1;

running\_processes = 1;

pflag[0] = 1;

int flag = 0;

while(running\_processes!=0)

{

flag = 0;

x++;

if(rt[pq[0]-1]>time\_quantum)

{

rt[pq[0]-1] = rt[pq[0]-1] - time\_quantum;

time = time + time\_quantum;

current = time;

}

else

{

time = time + rt[pq[0]-1];

rt[pq[0]-1] = 0;

flag = 1;

current = time;

tat[pq[0]-1] = time - at[pq[0]-1];

wt[pq[0]-1] = tat[pq[0]-1] - bt[pq[0]-1];

}

for(i=0;i<n;i++)

{

if(at[i] <= time && pflag[i]== 0)

{

pq[running\_processes] = i+1;

running\_processes = running\_processes + 1;

pflag[i] = 1;

}

}

if(x%2==0)

minptyinc(pty,n);

rearrange\_process\_queue(pq,rt,pty,n,running\_processes);

if(flag == 1)

running\_processes = running\_processes - 1;

}

printf("\n\nExecution Data:\n");

printf("|\tProcess\t|\tAT\t|\tBT\t| Priority |\tTAT\t|\tWT\t|\n");

for(i=0;i<n;i++)

{

printf("|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\n",i+1,at[i],bt[i],pty1[i],tat[i],wt[i]);

}

printf("\n\nAverage Waiting Time= %f\n",avg\_wait\_time(wt,n));

printf("Avg Turnaround Time = %f\n",avg\_turnaround\_time(tat,n));

return 0;

}

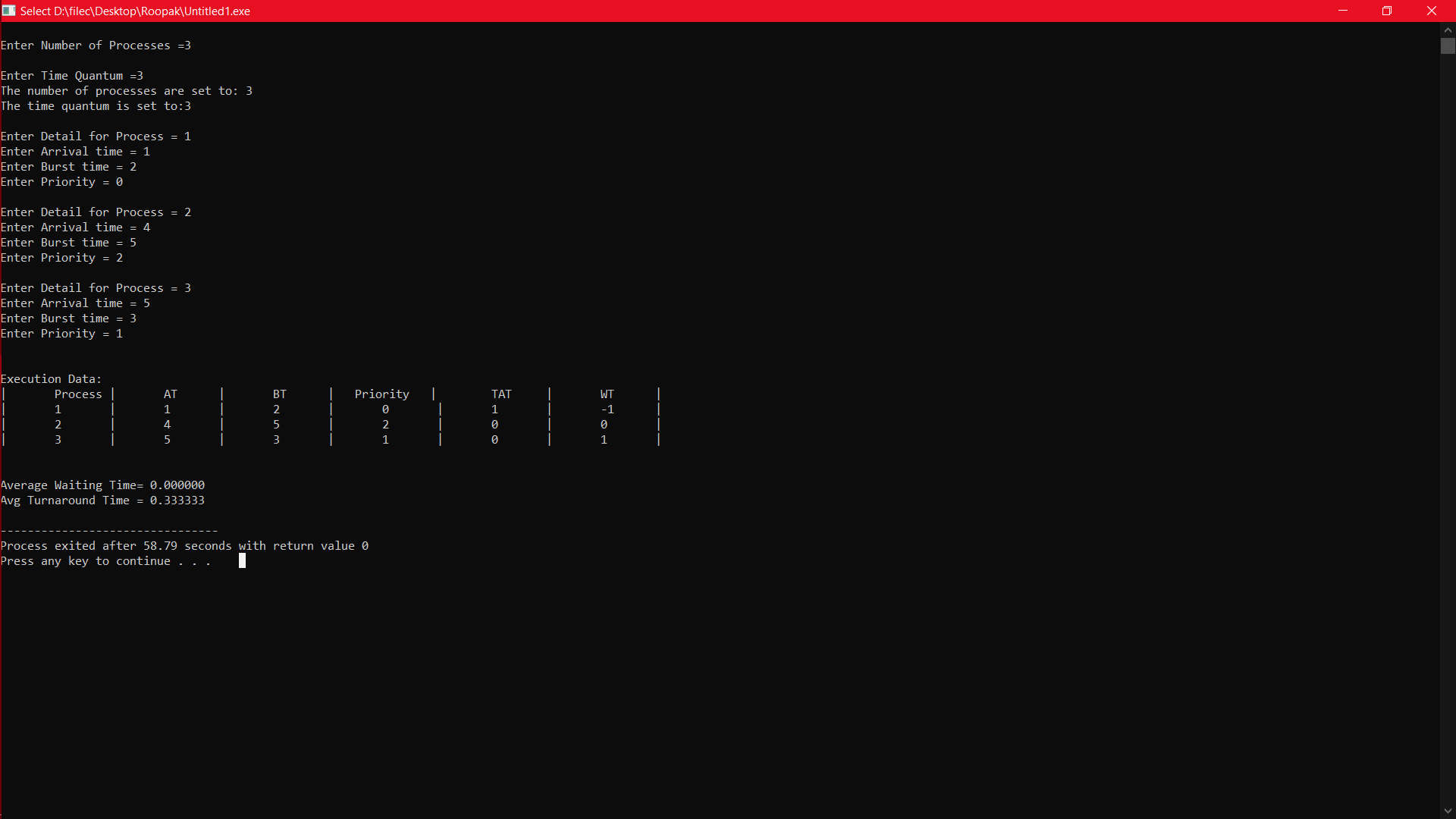
**Test cases:-**

**1.**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival time | Burst Time | Priority |
| P1 | 1 | 2 | 0 |
| P2 | 4 | 5 | 2 |
| P3 | 5 | 3 | 1 |

Average waiting time=0.00000

Avg Turnaround time=0.33333

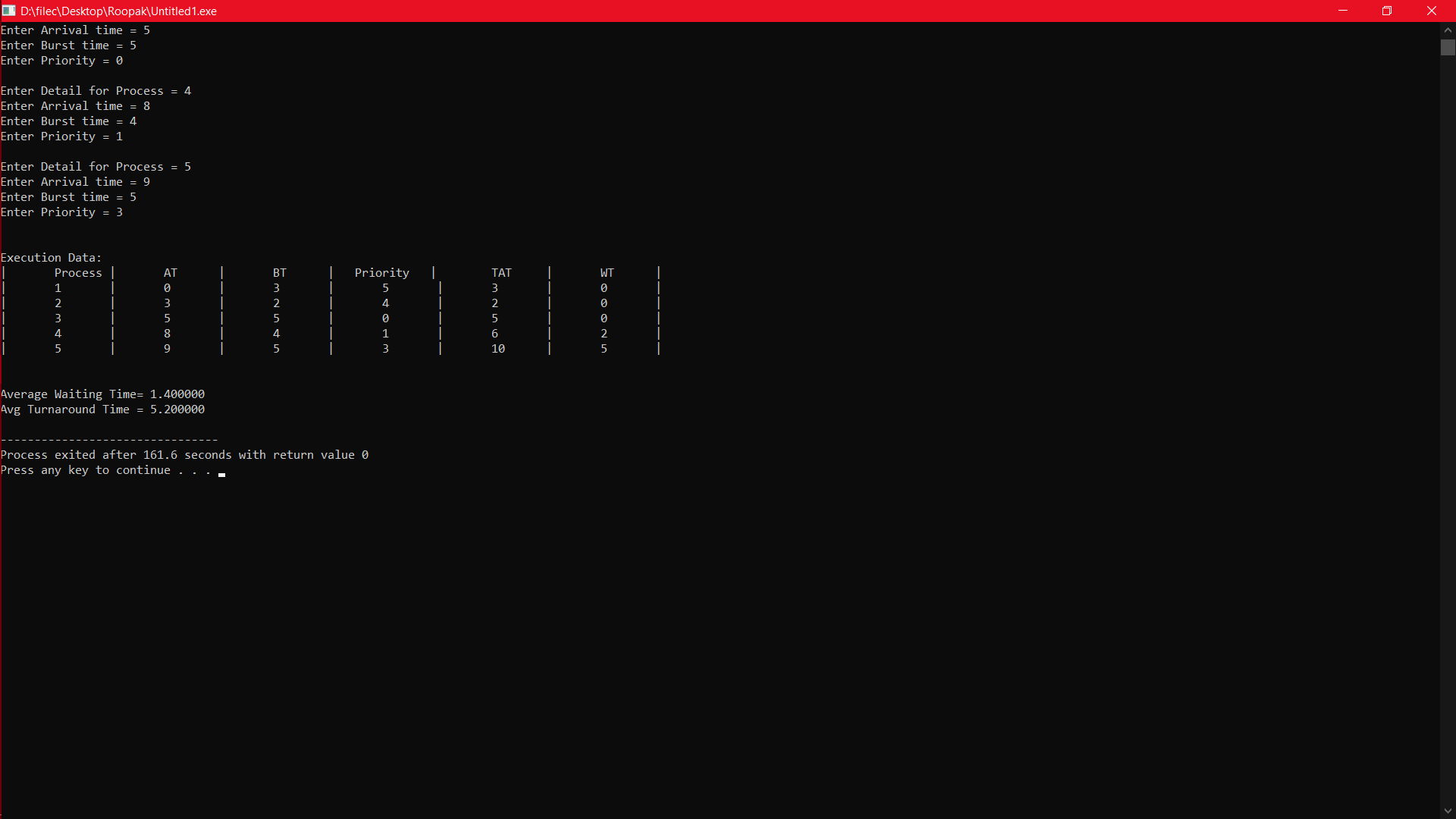


**2.**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival Time | Burst  Time | Priority |
| P1 | 0 | 3 | 5 |
| P2 | 3 | 2 | 4 |
| P3 | 5 | 5 | 0 |
| P4 | 8 | 4 | 1 |
| P5 | 9 | 5 | 3 |

Average Waiting Time= 1.400000

Average Turnaround Time = 5.200000



**3.**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival  Time | Burst  Time | Priority |
| P1 | 2 | 3 | 1 |
| P2 | 5 | 5 | 0 |
| P3 | 7 | 5 | 1 |
| P3 | 9 | 2 | 4 |

Average Waiting Time= 0.250000

Avg Turnaround Time = 0.250000



**Constraints:**

* If slicing time of OS is low, the processor output will be reduced.
* This method spends more time on context switching.
* Its performance heavily depends on time quantum.
* Priorities can not be set for the processes.
* Round-robin scheduling doesn’t give special priority to more important tasks.
* Decreases comprehension.
* Lower time quantum results in higher context switching overhead in the system.
* Finding a correct time quantum is a quite difficult task in this scheduling.

**Worst case latency:-**

This term is used for the maximum time taken for execution of all tasks

* dt = Denote detection time when a task is brought into the list
* st = Denote switching time from one task to another
* Formula: T(wors)t = {(dti+ sti + eti ), + (dti+ sti + eti )2 +...+ (dti+ sti + eti )N., + (dti+ sti + eti + eti) N} + t(ISR) t,SR = sum of all execution time,et = Denote task execution time