**STUDENT NAME**: Shrey Gupta

**STUDENT ID**: 11703591

**EMAIL ADDRESS**: shrey018@gmail.com

**GITHUB LINK**:

**DESCRIPTION:**

Algo used: - Round Robin and Priority Scheduling(Preemptive).

**ALGORITHM:**

Implemented Fixed priority preemptive Scheduling for level 1, and Used Round Robin Scheduling for level 2.

**CONSTRAINTS:**

The code must follow the working of given scheduling.

**CODE COMPLEXITY:**

Time complexity for Round Robin is **O(1),** and Average Worst time complexity for Fixed priority preemptive is **O(n).**

**NO. OF REVISIONS ON GITHUB:**

**Yes**, I do have 5 Revisions on GitHub.

**QUESTION :**

Ques. 7. Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running , another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2..

All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty. 2. Priority of Queue 2 has lower priority than in Queue 1.

**DESCRIPTION: -**

This Project contains implementation of Preemptive Round Robin Schedulingusing burst time and arrival time and this also implements the Fixed priority preemptive scheduling in which we can inputs from user and then arrange them according to the order.

**1.Round Robin Scheduling:**

Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a **quantum**. Once a process is executed for a given time period, it is preempted and other process executes for a given time period. Context switching is used to save states of preempted processes.

* 1. **Burst Time**

The time required by a process to CPU execution.

* 1. **Arrival time**

The time at which the process arrive in ready queue.

* 1. **Completion Time**

The time which the process complete its execution.

* 1. **Turnaround Time**

The time difference between completion time and arrival time.

Turnaround time: Completion Time – Arrival Time

* 1. **Waiting Time**

The time difference between turnaround time and burst time.

Waiting time: Turnaround time-Burst time

**CODE :**

#include<conio.h>

#include<stdlib.h>

#include<stdio.h>

#include<limits.h>

#include<unistd.h>

int p;

int burst[10],arrival[10],priority[10];

int process[10];

int around[10],waiting[10],g[10]={0},e[10];

int w,t;

int b[10];

int sg(int checker[],int i)

{

int j=0;

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

{

if(e[checker[j]]==0)

break;

}

return j;

}

int show()

{ system("cls");

sleep(3);

printf("\n\n\t\t\t================================================\n");

printf("\n\n\t\t\tProcess\t\t| Turnaround Time | Waiting Time\n\n");

printf("\t\t\t================================================\n\n");

int i=0;

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

{

printf("\t\t\tProcess[%d]\t|\t%d\t |\t%d\n",process[i],around[i],waiting[i]);

printf("\n\t\t\t================================================\n");

}

printf("\n\t\t\tAverage Waiting Time= %f",w\*1.0/p);

printf("\n\t\t\tAverage Turnaround Time = %f",t\*1.0/p);

printf("\n\n\t\t\t================================================\n\n");

}

int kg(int c,int b)

{

int i;

i=-1;

int j;

if(b!=-1)

{

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

{

if(arrival[j]==arrival[b])

{

continue;

}

else

{if(arrival[j]==c)

{

return j;

}

}

}

}

else

{

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

{

if(arrival[j]==c)

{

return j;

}

}

}

return i;

}

void prioritysort()

{

int i=0,j=0,k=p-1,max,temp=-1;

while(j<p)

{

max=-1;

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

{

if(max<=priority[i] && g[i]==0)

{

max=priority[i];

b[k]=i;

temp=i;

}

}

g[temp]=1;

j++;

k--;

}

}

int min(int time)

{

int i,j=-1;

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

{

if(arrival[i]==time && g[i] ==0)

{

j=i;

break;

}

}

if(j!=-1)

{

g[j]=1;

}

return j;

}

int main()

{

system("color 87");

printf("\n\n\n\n\n\t\t\t\t==========================================\n\n");

printf("\t\t\t\t\tShrey Gupta\n\t\t\t\t\tRegd No.: 11703591\n\t\t\t\t\tLovely Professional Univesity\n");

printf("\n\t\t\t\t==========================================\n");

sleep(10);

system("cls");

int i=0,j,temp,temp1;

system("color 07");

printf("enter no of processes:-");

scanf("%d",&p);

int fk[p];

int Q[p];

int Q2=0;

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

{

process[i]=i;

}

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

{

printf("enter burst time of process %d :-",i);

scanf("%d",&burst[i]);

fk[i]=burst[i];

printf("enter arrival time of process %d :-",i);

scanf("%d",&arrival[i]);

printf("enter priority of process %d :-",i);

scanf("%d",&priority[i]);

}

int timeline=0;

int remaining = p;

int check[10];

int k=0;

prioritysort();

int index;

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

g[i]=0;

i=0;

while(remaining!=0)

{

while((k = min(timeline))!=-1)

{

check[i]=k;

i++;

}

if(i==0 || sg(check,i)==i)

{

timeline++;

continue;

}

int index= INT\_MAX;

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

{

if(e[check[j]]!=1)

{

for(k=0;k<p;k++)

{

if(check[j]==b[k])

{

if(k<index)

{

index=k;

break;

}

}

}

}

}

if(index==INT\_MAX)

continue;

else{

temp1=b[index];

index=temp1;

temp=fk[index];

}

for(j=0;j<=temp&&(kg(timeline,index))==-1&& fk[index]>0;j++)

{

fk[index]--;

timeline++;

}

if(fk[index]>0)

{

while((k = min(timeline))!=-1)

{

check[i]=k;

i++;

while(priority[k]>priority[index]&&kg(timeline,k)==-1&&fk[index]!=0)

{ fk[index]--;

timeline++;

if(kg(timeline,-1)!=-1)

break;

}

}

if(priority[kg(timeline,-1)]<priority[index]&&kg(timeline,-1)!=-1)

{

Q[Q2]=index;

Q2++;

remaining--;

e[index]=1;

}

}

if(fk[index]==0)

{

waiting[index]=timeline-arrival[index]-burst[index];

w=w+waiting[index];

around[index]=timeline-arrival[index];

t=t+around[index];

e[index]=1;

remaining--;

}

}

int timeq=2;

remaining = Q2;

i=0;

int checking [10]={0};

while(remaining!=0)

{

if(fk[Q[i]]<=timeq && fk[Q[i]]>0)

{

timeline+=fk[Q[i]];

fk[Q[i]]=0;

}

else if(fk[Q[i]]>0)

{

fk[Q[i]]-=timeq;

timeline+=timeq;

}

if(fk[Q[i]]==0&&check[Q[i]]==0)

{

remaining--;

waiting[Q[i]]+=timeline-arrival[Q[i]]-burst[Q[i]];

w=w+waiting[Q[i]];

around[Q[i]]+=timeline-arrival[Q[i]];

t=t+around[Q[i]];

check[Q[i]]=1;

}

if(i==Q2-1)

i=0;

else

i++;

}

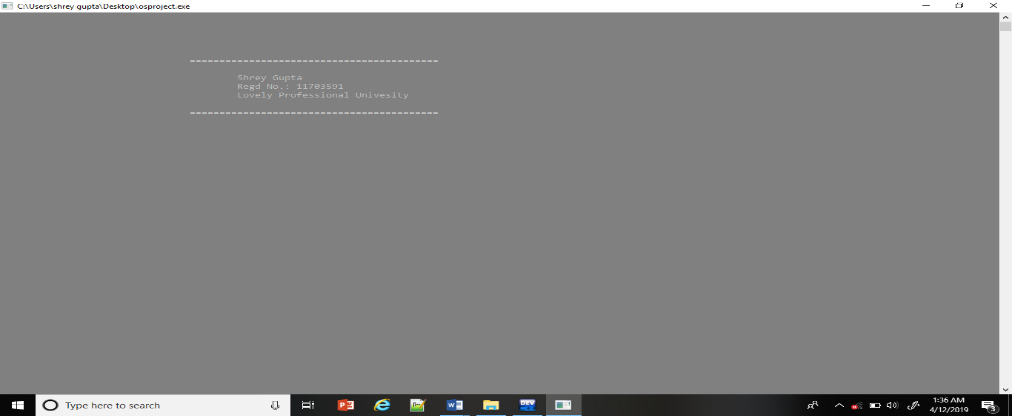
show();

getch();

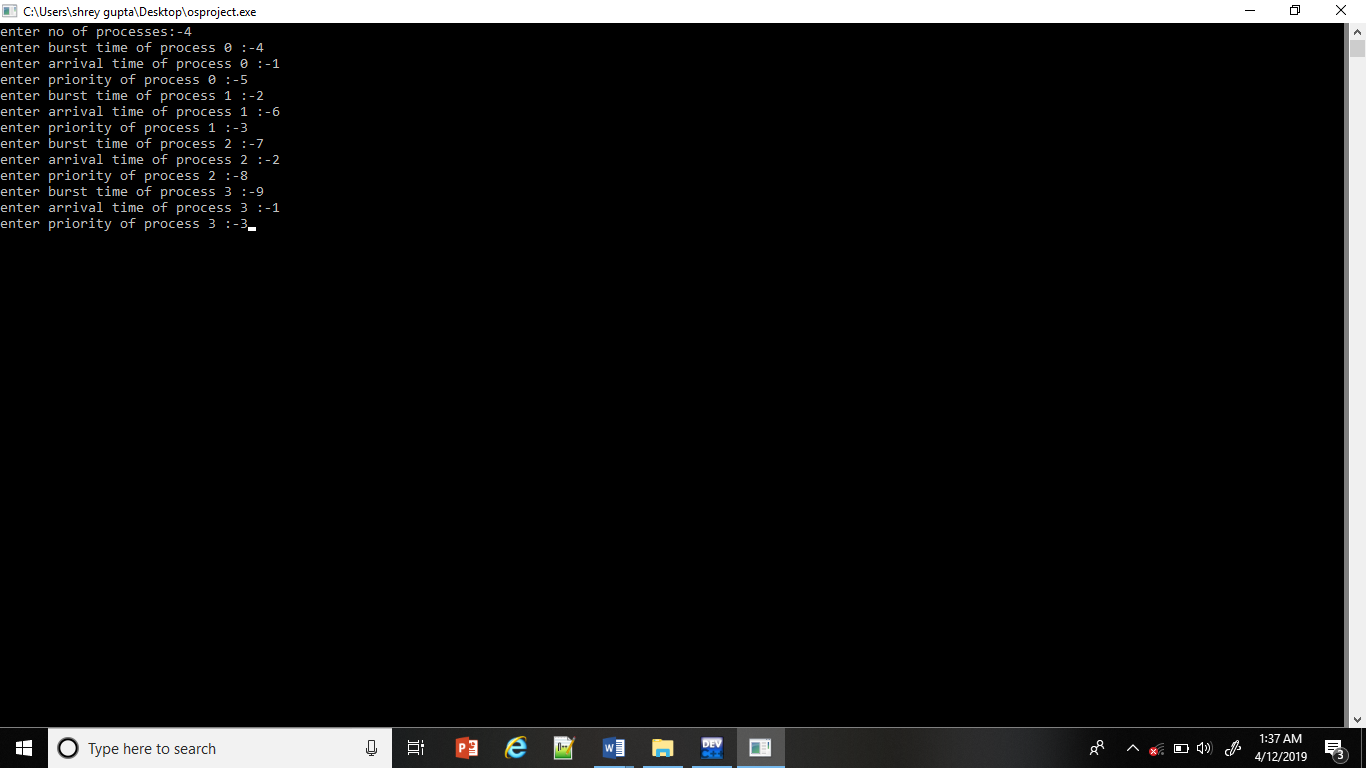
}

**OUTPUT OF QUESTION: SNAPSHOT**

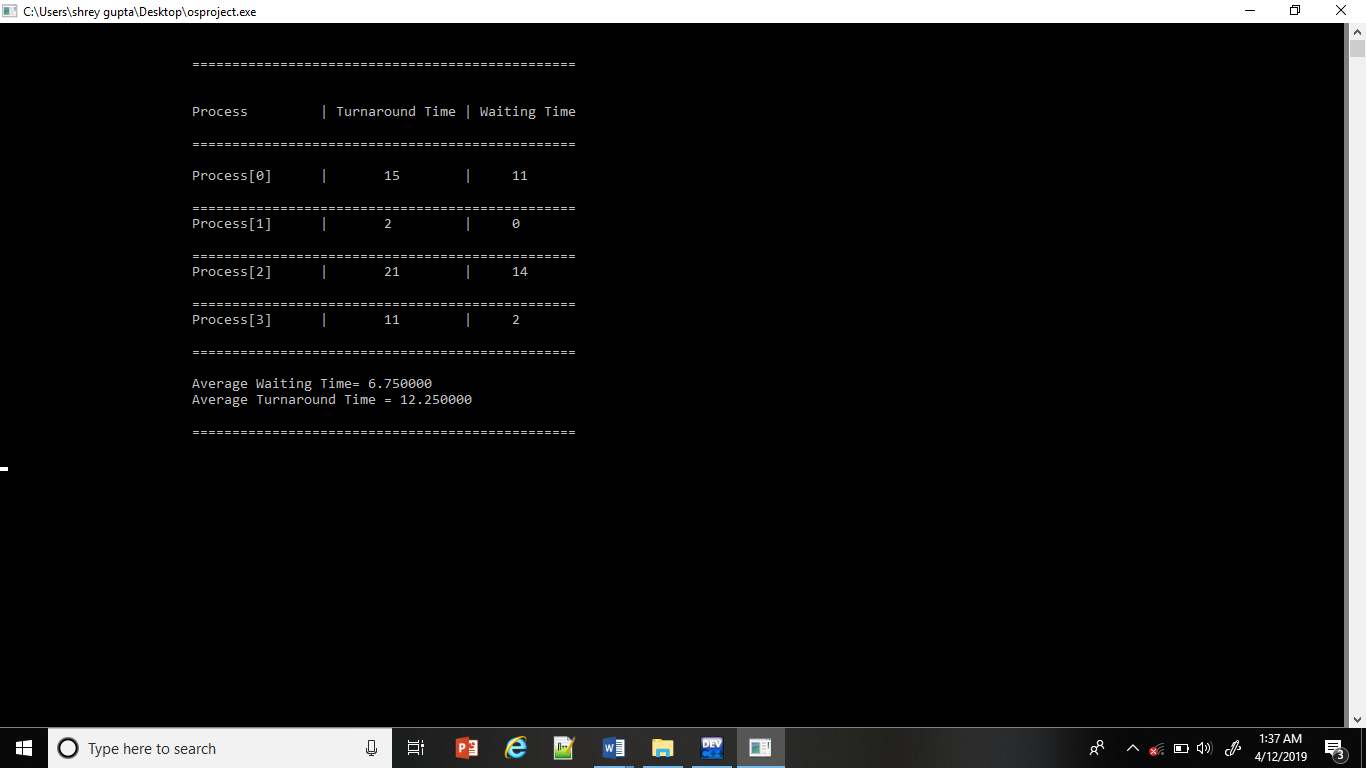
**1)**

****

**2)**

****

**3)**

****