Student Name: Hitesh Sangwan Registration No: 11714086

Student ID: 24 Section : K17KV

Email Address: hiteshsangwan0567@gmail.com

GitHub Link: https://github.com/HiteshSangwan0567/OS-Project

Code: CPU schedules N processes which arrive at different time intervals and each process is allocated the CPU for 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.

Problem Explained in operating System Concepts:

Operating system is an essential part of any computer system. CPU scheduling is the basic of multi-programmed operating systems. The operating system switches the CPU among processes to make the system more productive. To achieve this CPU must be as busy as possible; the CPU time should not be wasted because it affects the throughput of the system. In general, it can be said that CPU must execute some process all the time. To increase the performance the processes should be scheduled very efficiently. The scheduler deals with this. Whenever a CPU is idle, it fetches a process from memory and the dispatcher gives the control over CPU to the selected process by scheduler. The scheduler selects the process from memory by using CPU scheduling algorithms. The round robin scheduling algorithm works on the basic of a time quantum running parallel to the processes, regularly stopping them after a certain interval of time. This question deals with situation of high no of context switches, and how could we control them. We can use numerical priority for every process. This results in a higher priority being given to a smaller process, thus resulting in a yet smaller turn around time and waiting time. The proposed algorithm and code will take a numerical Quantum time T, and using the concept of round robin algorithm, will try to merge it with priority-based algorithm.

In the case of a priority-based algorithm the process with higher priority will come before the other processes and the process with lower priority will be handled later.

This problem tackles the main problem of round robin algorithm and make maximum utilization of CPU, making it more efficient and logically trustworthy to use in the future Operating Systems.

Proposed Algorithm for the Problem:

Step 1: begin

Step 2: initialize q=0,count=1, awt=0, att=0
,bt[100],wt[100],tat[100],b[100],time,remain,flag=0;

Step 3: read number of processes

Step 4: for each process i=1 to ///Complexity O(n) ……Every step is traversed once……..

4.1: read burst time

4.2: read process ID

4.3: read priority

4.2: End for

Step 5: calculate time quantum T.Q(Take from user)

Step 6: sort the processes in ascending order of their burst time, give high priority for short process

Step 7: while count = 1

7.1: count=0;

7.2: for each process i=1 to n

7.2.1: if bt[i]>tq then go to

7.2.1.1 else go to7.2.2

7.2.1.1: bt[i]=bt[i]-tq;

7.2.1.2: increment cst and count=1;

7.2.2: if bt[i] !=0 then goto

7.2.2.1else go to 7 7.2.2.1: bt[i]=0;

7.2.2.2: increment cst and count=1;

7.3 end for and while

Step 8: for each process i=1 to n find waiting time, turnaround time, AWT and ATT.

Step 9: Print the result

Step10: Stop.

Proposed Code Snippet with Complexity defined in C language:

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int bt[100],wt[100],tat[100],b[100],time,remain,flag=0;////Initialization of variables//////
 int i,j, q=0,n,count=1, a, n1, tq , pr[100], temp, cst, temp1, rt[100], at[100]; ////Initialization of variables//////
 char pid[100]; ////Initialization of variables//////
 float twt=0,tTT=0,avgwt,avgtt; ////Initialization of variables//////
 //clrscr();////To clear the screen after every execution/////
 printf("Enter total number of process : ");////Take the no of  Process ID from here////
 scanf("%d",&n);
 printf("Enter the time Quantum");//// Take the Time Quantum from the user/////
 scanf("%d", &tq);
```

```
 remain=n;


 for(i=0;i<n;i++)(((Complexity of the for function : O(n) , traversal is done once))))
 {
  printf("\n Enter the process ID %d",i+1);////Take the process ID names//////
  scanf("%s",&pid[i]);
  printf("\nEnter the burst time of %d process : ",i+1);///// Take the process Burst Times/////
  scanf("%d",&bt[i]);
  printf("\nEnter the Priority of process %d", i+1);//////Enter the Priority//////
  scanf("%d",&pr[i]);


  b[i]= bt[i];////Saving the value of burst Time/////
}
  int pos;
  for(i=0;i<n;i++)…….(1)///(((Complexity= O(n*n)……Traversed 2 times because of two
times for loop)))////……..Also the code for the priority……..///////
    {
       pos=i;
     for(j=i+1;j<n;j++)……(2)
       {
          if(pr[j]<pr[pos])
              pos=j;
       }


       temp=pr[i];
       pr[i]=pr[pos];
       pr[pos]=temp;


       temp=bt[i];
```

```
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=pid[i];
      pid[i]=pid[pos];
      pid[pos]=temp;
   }
i=0;
 for(time=0;count!=n;time++)…….Code to calculate total turn around Time, Average Turn around time, Waiting time , Average Waiting Time……Time Quantum.
 {
  while(bt[i] == 0)
  {
    i=(i+1)%n;
  }
  bt[i]--;
  if(bt[i]==0)
  {
    tat[i] = time+1;
    count++;
  }
  i = (i+1)%n;
 }
 printf("\nprocess  burst  waitng  turnaround  ");
 for(i=0;i<n;i++)(((((Complexity of O(n)))))))((((One time Traversal)))))
 {  wt[i] = tat[i] - b[i];
   printf("\n  %d \t  %d \t  %d \t  %d",i+1,b[i],wt[i],tat[i]); /////Printing of Final Result/////
   twt = twt + wt[i];
   tTT = tTT + tat[i];
 }
```

```
 printf("\n  %d  %f  %f",n,twt,tTT); /////Printing of Final Result/////
 avgwt = twt / n;
 avgtt = tTT / n;
 printf("\nAverage waiting time is %f",avgwt);/////Printing of Final Result/////
 printf("\nAverage turnaround time is %f ",avgtt); /////Printing of Final Result/////
 getch();
}
```

Description(Purpose of Use):

Constraints:

1. Process allocation, Time Burst and Priority Allocation::

```
for(i=0;i<n;i++)(((Complexity of the for function : O(n) , traversal is done once))))
 {
  printf("\n Enter the process ID %d",i+1);////Take the process ID names//////
  scanf("%s",&pid[i]);
  printf("\nEnter the burst time of %d process : ",i+1);///// Take the process Burst Times/////
  scanf("%d",&bt[i]);
  printf("\nEnter the Priority of process %d", i+1);//////Enter the Priority//////
  scanf("%d",&pr[i]);

  b[i]= bt[i];////Saving the value of burst Time/////
}
```

2. Priority based classification(Sorting the Process ID, Priority and Time Burst in ascending order:

```
     pos=i;
    for(j=i+1;j<n;j++)……(2)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }
```

```
    temp=pr[i];
    pr[i]=pr[pos];
    pr[pos]=temp;

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=pid[i];
    pid[i]=pid[pos];
    pid[pos]=temp;
```

3. Calculating final Result, Turn Around Time, Average Time, Waiting Time etc.

```
i=0;
 for(time=0;count!=n;time++)…….Code to calculate total turn around Time, Average Turn around time, Waiting time , Average Waiting Time……Time Quantum.
 {
   while(bt[i] == 0)
   {
     i=(i+1)%n;
   }
   bt[i]--;
   if(bt[i]==0)
   {
     tat[i] = time+1;
     count++;
   }
   i = (i+1)%n;
```

```
 }
 printf("\nprocess  burst  waitng  turnaround  ");
 for(i=0;i<n;i++)(((((Complexity of O(n))))))((((One time Traversal))))
 {  wt[i] = tat[i] - b[i];
   printf("\n  %d \t   %d \t    %d \t   %d",i+1,b[i],wt[i],tat[i]); /////Printing of Final
Result/////
   twt = twt + wt[i];
   tTT = tTT + tat[i];
 }
 printf("\n  %d  %f  %f",n,twt,tTT); /////Printing of Final Result/////
 avgwt = twt / n;
 avgtt = tTT / n;
 printf("\nAverage waiting time is %f",avgwt);/////Printing of Final Result/////
 printf("\nAverage turnaround time is %f ",avgtt); /////Printing of Final Result/////
 getch();
}
```

Additional Program in c:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  char p[10][5],temp[5];
  int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
  float avgwt;
 printf("enter no of processes:");
 scanf("%d",&n);
  for(i=0;i<n;i++)
  {
  printf("enter process%d name:",i+1);
 scanf("%s",&p[i]);
  printf("enter process time:");
 scanf("%d",&pt[i]);
  printf("enter priority:");
```

```
 scanf("%d",&pr[i]);
 }
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
  if(pr[i]>pr[j])
{
  temp1=pr[i];
pr[i]=pr[j];
 pr[j]=temp1;
 temp1=pt[i];
 pt[i]=pt[j];
 pt[j]=temp1;
strcpy(temp,p[i]);
strcpy(p[i],p[j]);
 strcpy(p[j],temp);
 }
 }
 }
wt[0]=0;
 for(i=1;i<n;i++)
 {
  wt[i]=wt[i-1]+et[i-1];
  totwt=totwt+wt[i];
  }
avgwt=(float)totwt/n;
printf("p_name\t p_time\t priority\t w_time\n");
for(i=0;i<n;i++)
{
   printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
  }
 printf("total waiting time=%d\n avg waiting time=%f",tot,avg);
 getch();
 }
```

Boundary Conditions of the Implemented Code:

1. The boundary conditions of the implemented code:
   a. The n is the number of the processes defines : The value of i cannot surpass this boundary condition:

```
printf("Enter total number of process : ");////Take the no of  Process ID from here////
 scanf("%d",&n);
 printf("Enter the time Quantum");//// Take the Time Quantum from the user/////
 scanf("%d", &tq);

 remain=n;

 for(i=0;i<n;i++)(((Complexity of the for function : O(n) , traversal is done once))))
 {
  printf("\n Enter the process ID %d",i+1);////Take the process ID names//////
  scanf("%s",&pid[i]);
  printf("\nEnter the burst time of %d process : ",i+1);///// Take the process Burst Times/////
  scanf("%d",&bt[i]);
  printf("\nEnter the Priority of process %d", i+1);//////Enter the Priority//////
  scanf("%d",&pr[i]);
```

b. The proposed time cannot be greater than the whole cpu time:

```
for(time=0;count!=n;time++)
```

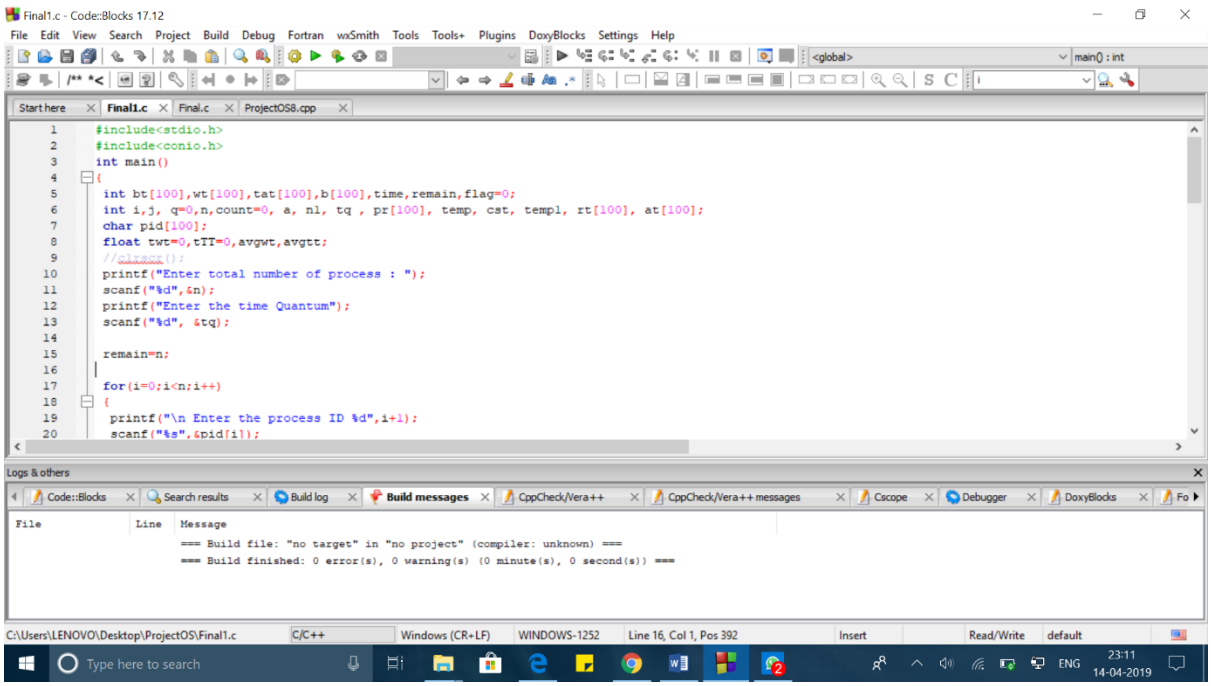
c. The proposed no of processes cannot be exceeded in order to calculate total time and average waiting time:

```
for(i=0;i<n;i++)((((((Complexity of O(n))))))((((One time Traversal))))))
 {  wt[i] = tat[i] - b[i];
    printf("\n   %d \t    %d \t     %d \t    %d",i+1,b[i],wt[i],tat[i]); /////Printing of Final
Result/////
    twt = twt + wt[i];
    tTT = tTT + tat[i];
 }
 printf("\n  %d  %f  %f",n,twt,tTT); /////Printing of Final Result/////
 avgwt = twt / n;
```

```
avgtt = tTT / n;
printf("\nAverage waiting time is %f",avgwt);/////Printing of Final Result/////
printf("\nAverage turnaround time is %f ",avgtt); /////Printing of Final Result/////
getch();
}
```

Test Case: (Using 1st Program):

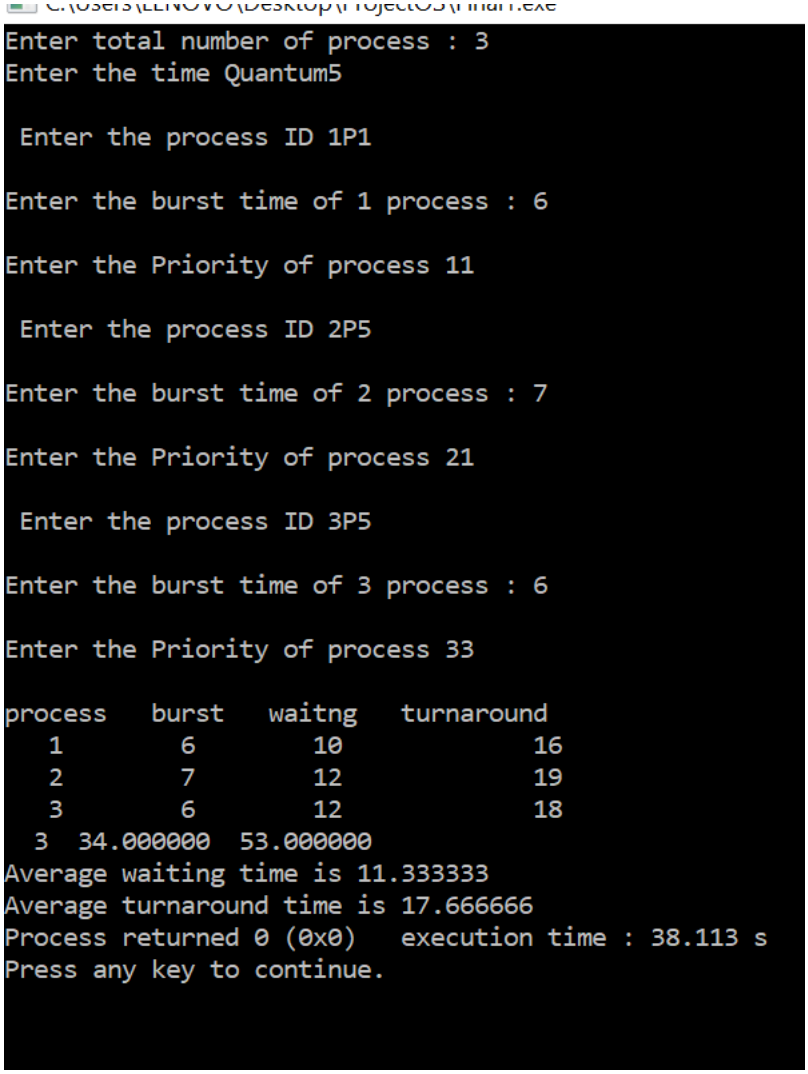
Successful Compilation of the problem:



1. Test Case 1……

```
Enter total number of process : 4
Enter the time Quantum4

 Enter the process ID 1P1

Enter the burst time of 1 process : 4

Enter the Priority of process 11

 Enter the process ID 2P2

Enter the burst time of 2 process : 6

Enter the Priority of process 22

 Enter the process ID 3P3

Enter the burst time of 3 process : 7

Enter the Priority of process 33

 Enter the process ID 4P4

Enter the burst time of 4 process : 8

Enter the Priority of process 45

process   burst   waitng   turnaround
   1        4        9      13
   2        6        14             20
   3        7        16             23
   4        8        17             25
  4  56.000000  81.000000
Average waiting time is 14.000000
Average turnaround time is 20.250000
```

Test case 2:



GitHub: The complete algorithm , with test cases , results , source code in 4 different languages can be found on the github repository.
https://github.com/HiteshSangwan0567/OS-Project