Name:P.Saikirankumarreddy
Roll Number:12
Reg.no:11807870
Section:K18NS
Gmail:saikiranreddypaleti@gmail.com
Github Link:

Problem:

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 the numerical priority,with a higher number indicating a higher relative priority.in addition to the processes one task has priority 0.The length of the time quantam is T units,Where T is the custom time considered as time quantam for processing.if a processes 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 .

Solution:

- Round robin is the preemptive scheduling algorithm.
- Each process is provided a fix time to execute is called time quantam.
- Once a process is executed for a given time period ,it is preempted and other processs executes for a given time period .
- Context Switching is used to save the states of preempted process.
- Ready Queue and Running Queue are used.

Completion time:Time at which process completes its execution.
Turn Around Time:Time differenece between completion time and arrival time.
Turn around time=completion time-arrival time
Waiting time:Time difference between turn around time and burst time.
Waiting Time=Turn around time-Burst time

Algorithm:

- Start the process
- Enter the number of process to be executed
- Enter the time quantam .
- Then enter the arrivaltime of the process .
- Then enter the priority and burst time of the process
- Repeat the same steps for the remaining process.
- And then find the waiting time of process(n-1)+time difference in getting the CPU from process(n-1).
- Exexute the process according to the priority number if the process doesnot complete with in the time quantam it should be added to last of the queue by lowering down the priority.
- Compute waiting time and turn around time.

Complexity:

The complexity of the round robin algorithm is O(1).it is easy to realize and is suitable to use in high speed networks.One of the most classical is DRR scheduling algorithm.But DRR algorithm has some shortcomings that is delay charectaristic is not very ideal and its output burst is big.

Advantages:

Starvation which is never occured in ROUND ROBIN ALGORITHM scheduling.All the process is just using the one cpu.One of the main advantage is this scheduling can work on any os.

Disadvantages:

Longer process may starve.

Performance is heavily dependent on the time quantam.

No idea of priority.

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++)
	{
```

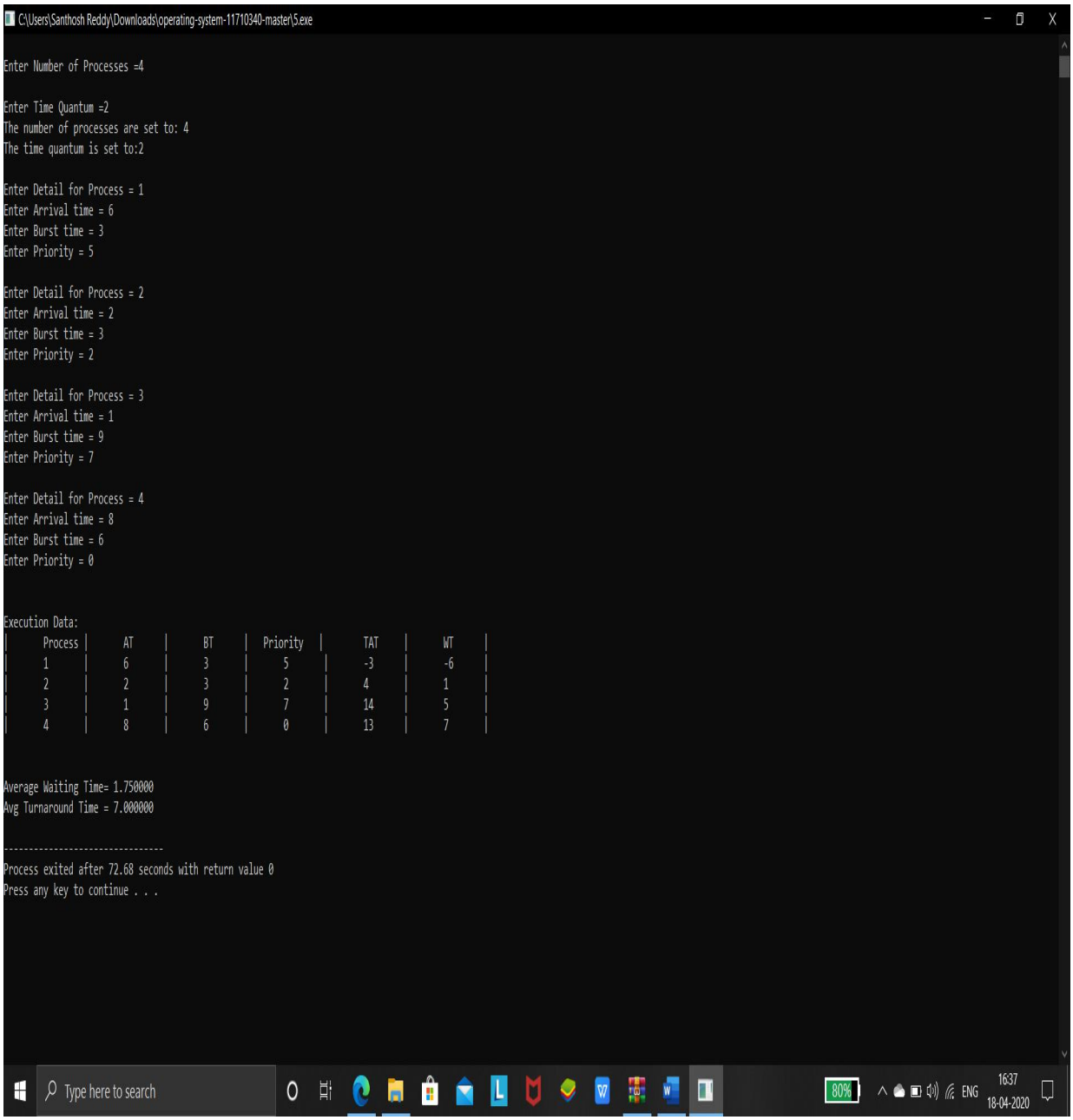
```
            pq[i] = pq[i+1];
        }
        pq[running_processes-1] = temp;

        }
        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;
}
```

Output:


C:\Users\Santhosh Reddy\Downloads\operating-system-11710340-master\5.exe
Enter Number of Processes =4
Enter Time Quantum =2
The number of processes are set to: 4
The time quantum is set to:2
Enter Detail for Process = 1
Enter Arrival time = 6
Enter Burst time = 3
Enter Priority = 5
Enter Detail for Process = 2
Enter Arrival time = 2
Enter Burst time = 3
Enter Priority = 2
Enter Detail for Process = 3
Enter Arrival time = 1
Enter Burst time = 9
Enter Priority = 7
Enter Detail for Process = 4
Enter Arrival time = 8
Enter Burst time = 6
Enter Priority = 0
Execution Data:
Process	AT	BT	Priority	TAT	WT
1	6	3	5	-3	-6
2	2	3	2	4	1
3	1	9	7	14	5
4	8	6	0	13	7
Average Waiting Time= 1.750000
Avg Turnaround Time = 7.000000
Process exited after 72.68 seconds with return value 0
Press any key to continue . . .
Type here to search
80%
ENG
1637
18-04-2020