**EXERCISE-2**

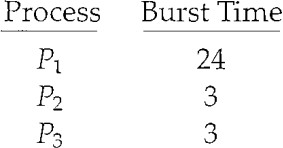
**Round Robin** CPU scheduling algorithm

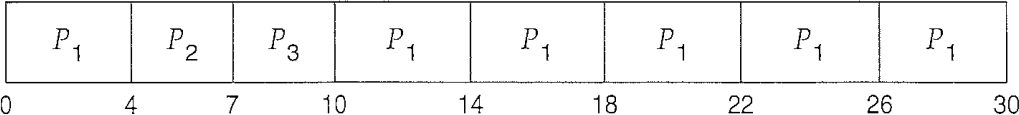
**Aim:**

To implement **Round Robin** CPU scheduling algorithm.

**Description:**

## Round Robin CPU scheduling algorithm:

* It is a preemptive scheduling algorithm.
* The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
* It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
* A small unit of time, called a time quantum or time slice, is defined.
* The ready queue is treated as a circular queue.
* The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
* The average waiting time under the RR policy is often long.
* Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:
* If we use a time quantum of 4 milliseconds
* Then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2. Process P2 does not need 4 milliseconds, so it quits before its time quantum expires.
* The CPU is then given to the next process, process P3.
* Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum.
* The resulting RR schedule is as follows:



* P1 waits for 6millisconds (10- 4), P2 waits for 4 milliseconds, and P3 waits for 7 milliseconds.
* Thus, the average waiting time is 17/3 = 5.66 milliseconds.
* In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row (unless it is the only runnable process).
* If there are n. processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units.
* Each process must wait no longer than (n - 1) x q time units until its next time quantum.

**Program:**

#include<stdio.h>

struct process

{

char name[10];

int bt,ft,tat,wt,tft;

}p[10];

int main()

{

int i,j=0,k=0,n,d,timeslice,ttat=0,tbt=0,twt=0;

float avgwt,avgtat;

printf("enter number of processes:");

scanf("%d",&n);

printf("enter time slice:");

scanf("%d",&timeslice);

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

{

printf("enter name of process %d :",i+1);

scanf("%s",p[i].name);

printf("enter burst time of process %d :",i+1);

scanf("%d",&p[i].bt);

}

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

{

tbt=tbt+p[i].bt; p[i].tft=0;

p[i].wt=0;

}

while(j<tbt)

{

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

{

if(p[i].bt>0)

{

if(p[i].bt<=timeslice)

{

if(k==0&&i==0)

p[i].wt=0;

else

{

d=j-p[i].tft;

p[i].wt+=d;

}

j+=p[i].bt;

p[i].ft=j;

p[i].bt=0;

}

else

{

if(k==0&&i==0)

p[i].wt=0;

else

{

d=j-p[i].tft;

p[i].wt+=d;

}

j+=timeslice; p[i].bt-=timeslice; p[i].tft=j;

}

}

}

k++;

}

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

{

printf("\n\n");

printf("waiting time of process %s :%d\n",p[i].name,p[i].wt);

printf("turn around time of process %s :%d\n",p[i].name,p[i].ft);

printf("finish time of process %s :%d\n",p[i].name,p[i].ft);

}

printf("Processes Waiting Time Turn Around Time Finish Time\n");

printf("---------------------------------------------------------\n");

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

{

printf(" %s \t\t%d\t\t %d\t\t %d",p[i].name,p[i].wt,p[i].ft,p[i].ft);

printf("\n");

}

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

{

ttat+=p[i].ft;

twt+=p[i].wt;

}

avgwt=(float)twt/n; avgtat=(float)ttat/n;

printf("Total turn around time:%d\n",ttat);

printf("Total waiting time:%d\n",twt);

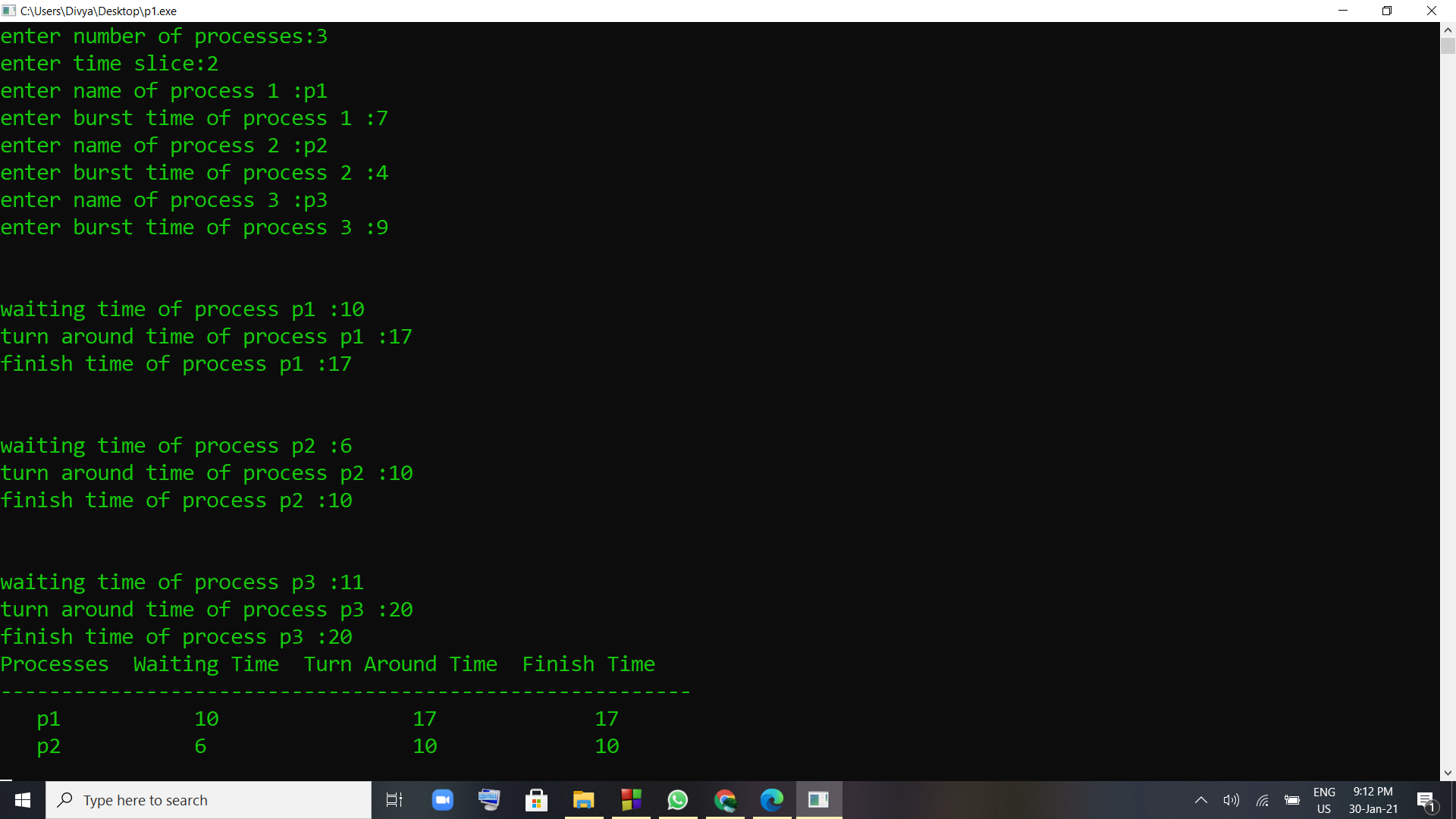
printf("Avg turn around time:%f\n",avgtat);

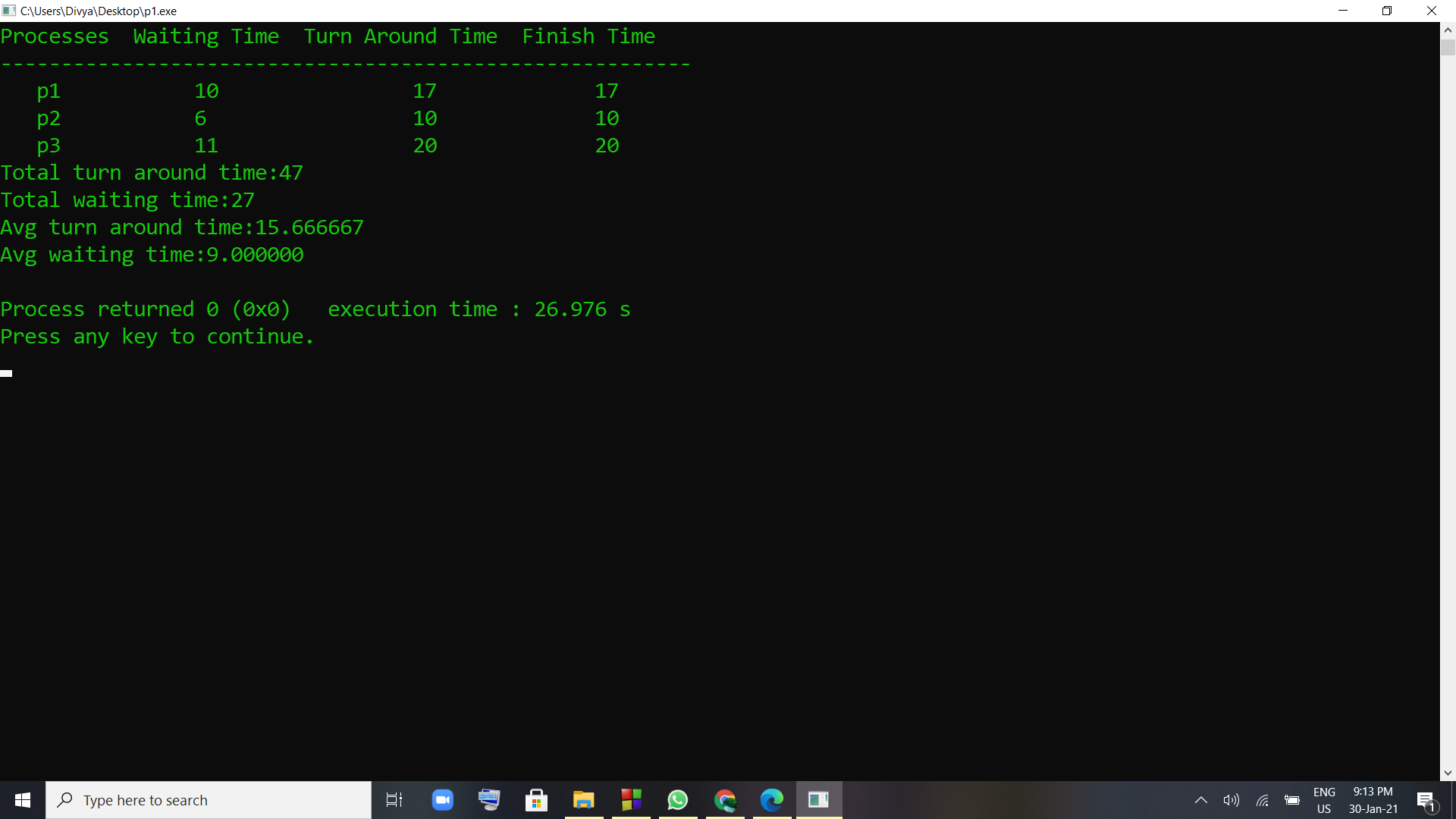
printf("Avg waiting time:%f\n",avgwt);

return 0;

}

**Output:**

****

****