PROCESS SCHEDULING

- CPU scheduling is used in multiprogrammed operating systems.
- By switching CPU among processes, efficiency of the system can be improved.
- Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.
- Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.

FIRST COME FIRST SERVE (FCFS):

- Process that comes first is processed first
- FCFS scheduling is non-preemptive
- Not efficient as it results in long average waiting time.
- Can result in starvation, if processes at beginning of the queue have long bursts.

SHORTEST JOB FIRST (SJF):

- Process that requires smallest burst time is processed first.
- SJF can be preemptive or non–preemptive
- When two processes require same amount of CPU utilization, FCFS is used to break the tie.
- Generally efficient as it results in minimal average waiting time.
- Can result in starvation, since long critical processes may not be processed.

PRIORITY:

- Process that has higher priority is processed first.
- Prioirty can be preemptive or non–preemptive
- When two processes have same priority, FCFS is used to break the tie.
- Can result in starvation, since low priority processes may not be processed.

ROUND ROBIN:

- All processes are processed one by one as they have arrived, but in rounds.
- Each process cannot take more than the time slice per round.
- Round robin is a fair preemptive scheduling algorithm.
- A process that is yet to complete in a round is preempted after the time slice and put at the end of thequeue.
- When a process is completely processed, it is removed from the queue.

EX.NO: 1a **FCFS SHEDULING**

DATE: 03.02.2023

AIM:

To schedule snapshot of processes queued according to FCFS (First Come First Serve) scheduling.

ALGORITHM:

1. Define an array of structure process with members process , burst time, waiting time and ttime.
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain burst time for each process.
4. The waiting time for first process is 0.
5. Compute waiting time and turnaround time for each process as:

a. waiting time (i+1) = waiting time (i) + burst time (i)

b. turnaround time (i) = waiting time (i) + burst time(i)

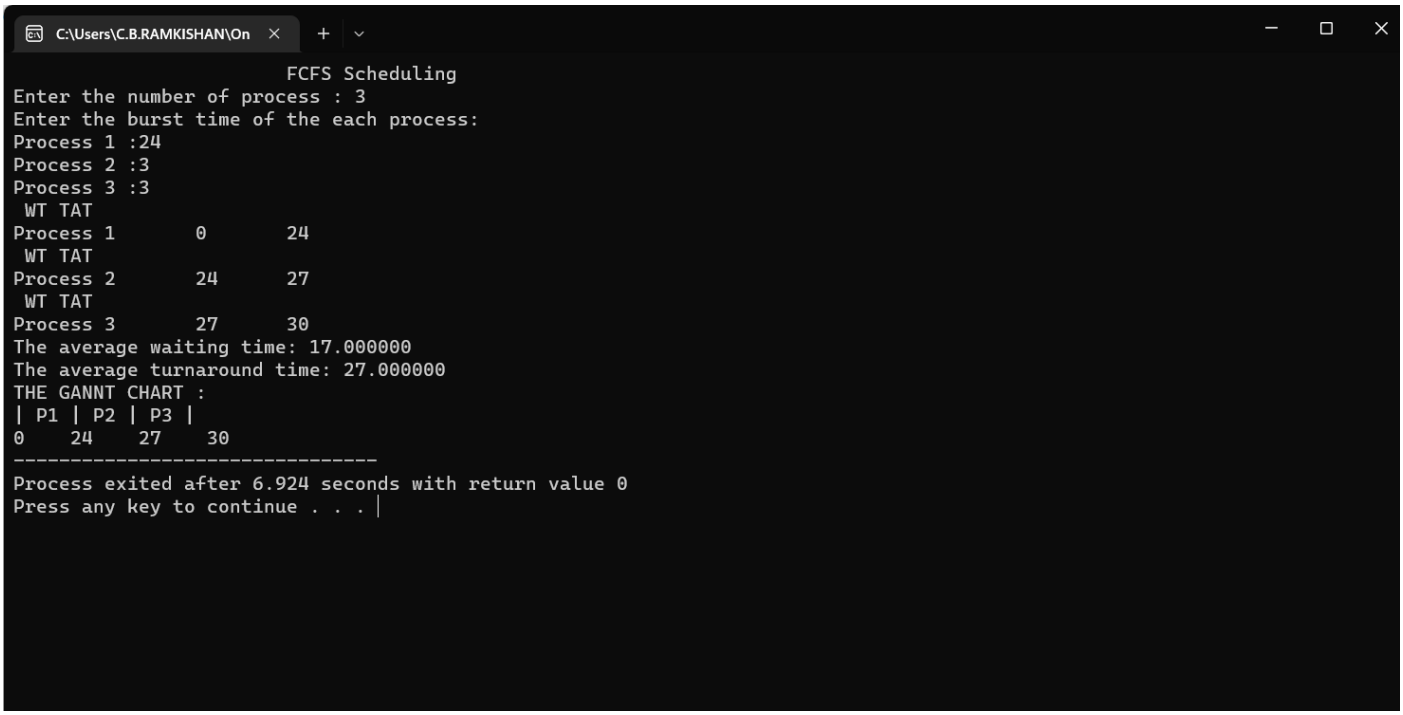
6. Compute average waiting time and average turnaround time
7. Display the burst time, turnaround time and waiting time for each process.
8. Display GANTT chart for the above scheduling
9. Display average waiting time and average turnaround time
10. Stop

CODE:

```
#include<stdio.h>
#include<conio.h>
//global variables
int n,wt[10],bt[10],tat[10];
//Calculating waiting time for each process
void waitingtime(){
        int i;
        wt[0]=0;
        for(i=1;i<n;i++)
                wt[i]=bt[i-1]+wt[i-1];
}
//Calculating turnaround time for each process
void turnaroundtime(){
        int i;
        for(i=0;i<n;i++)
                tat[i]=wt[i]+bt[i];
}
```

```
//Calculating the average waiting time and turnaround time
void average(){
	int twt=0,ttat=0;
	float awt,atat;
	int i;
	waitingtime();
	turnaroundtime();
	printf("Wating time and turnaround time for each process: \n");
	for(i=0;i<n;i++){
		printf("Process %d\t%d\t%d\n",i+1,wt[i],tat[i]);
		twt+= wt[i];
		ttat+=tat[i];
	}
	awt=(float)twt/n;
	atat=(float)ttat/n;
	printf("The average waiting time: %f\n",awt);
	printf("The average turnaround time: %f\n",atat);
}

//Dispalying the gantt chart
void ganttchart(){
	int i;
	printf("The Gantt Chart \n");
	printf("| ");
	for(i=0;i<n;i++)
		printf("P%d | ",i+1);
	printf("\n0   ");
	for(i=0;i<n;i++)
		printf("%d   ",tat[i]);
}

//Main function
int main(){
	int i;
	printf("\t\t\tFCFS Scheduling\n");
	printf("Enter the number of process : ");
	scanf("%d",&n);
	printf("Enter the burst time of the each process: \n");
	for(i=0;i<n;i++){
		printf("Process %d :",i+1);
		scanf("%d",&bt[i]);
	}
	average();
	ganttchart();
	return 0;
}
```

OUTPUT:



RESULT:

Thus Waiting time and Turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

EX NO: 1b **SJF SCHEDULING**

DATE:03.02.2023

AIM:

To schedule snapshot of processes queued according to SJF (Shortest Job First) scheduling.

ALGORITHM:

1. Define an array of structure process with members pid, btime, wtime & ttime.
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain btime for each process.
4. Sort the processes according to their btime in ascending order.
    a. If two process have same burst time, then FCFS is used to resolve the tie.
5. The waiting time for first process is 0.
6. Compute waiting time and turnaround time for each process as:
    a. Waiting time (i+1) = Waiting time (i) + burst time (i)
    b. Turnaround time (i) = Waiting time (i) + burst time (i)
7. Compute average waiting time and average turnaround time.
8. Display burst time, turnaround time and waiting time for each process.
9. Display GANTT chart for the above scheduling
10. Display average waiting time and average turnaround time.
11. Stop

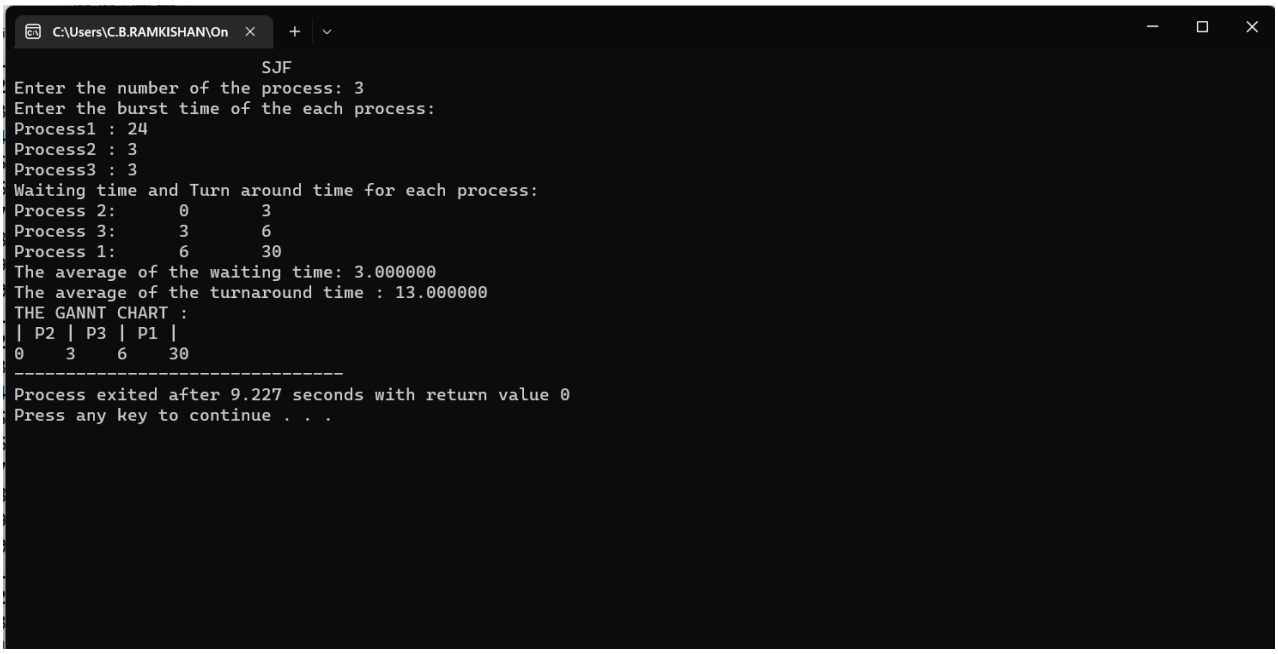
CODE:

```
#include<stdio.h>
#include<conio.h>

//global variable
int n,process[10],bt[10],wt[10],tat[10];

// Sort the burst time -- selection sort technique is used
void sortbt(){
        int i,j,pos,temp;
        for(i=0;i<n;i++){
                pos=i;
                for(j=i+1;j<n;j++){
```

```
                        if(bt[j]<bt[pos])
                                pos=j;
                }
                //burst time is swapped
                temp=bt[pos];
                bt[pos]=bt[i];
                bt[i]=temp;
                //Process number is swapped
                temp=process[pos];
                process[pos]=process[i];
                process[i]=temp;
        }
}

//Calculating the waiting time
void waitingtime(){
        int i;
        wt[0]=0;
        for(i=1;i<n;i++)
                wt[i]=bt[i-1]+wt[i-1];
}

//Calculating the turn around time
void turnaroundtime(){
        int i;
        for(i=0;i<n;i++)
                tat[i]=bt[i]+wt[i];
}

//Calculating the average waiting time and turnaround time
void average(){
        float awt,atat;
        int twt=0,ttat=0,i;
        sortbt();
        waitingtime();
        turnaroundtime();
        printf("Waiting time and Turn around time for each process: \n");
        for(i=0;i<n;i++){
                printf("Process %d:\t%d\t%d\n",process[i],wt[i],tat[i]);
                ttat+=tat[i];
                twt+=wt[i];
        }
        awt=(float)twt/n;
        atat=(float)ttat/n;
        printf("The average of the waiting time: %f\n",awt);
        printf("The average of the turnaround time : %f\n",atat);
}
//Displaying the gantt chart
void ganttchart(){
        int i;
        printf("| ");
```

```
        for(i=0;i<n;i++)
                printf("P%d | ",process[i]);
        printf("\n0    ");
        for(i=0;i<n;i++)
                printf("%d    ",tat[i]);
}

//Main function
void main(){
        int i;
        clrscr();
        printf("\t\t\tSJF\n");
        printf("Enter the number of the process: ");
        scanf("%d",&n);
        printf("Enter the burst time of the each process: \n");
        for(i=0;i<n;i++){
                printf("Process%d : ",i+1);
                scanf("%d",&bt[i]);
                process[i]=i+1;
        }
        average();
        ganttchart();
        getch();
}
```

OUTPUT:



Observation (20)	
Record(5)	
Total (25)	
Intial	

RESULT:

Thus Waiting time and Turnaround time for processes based on SJF scheduling was computed and the average waiting time was determined.

Ex No:2.a **Priority CPU Sheduling**

Date : 08.02.2023

AIM:

To schedule snapshot of processes queued according to Priority scheduling.

ALGORITHM:

1. Define an array of structure process with members pid, btime, pri, wtime & ttime.
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain btime and pri for each process.
4. Sort the processes according to their pri in ascending order.
a. If two process have same pri, then FCFS is used to resolve the tie.
5. The wtime for first process is 0.
6. Compute wtime and ttime for each process as:
a. wtimei+1 = wtimei + btimei
b. ttimei = wtimei + btimei
7. Compute average waiting time awat and average turn around time atur
8. Display the btime, pri, ttime and wtime for each process.
9. Display GANTT chart for the above scheduling
10. Display awat and atur
11. Stop.

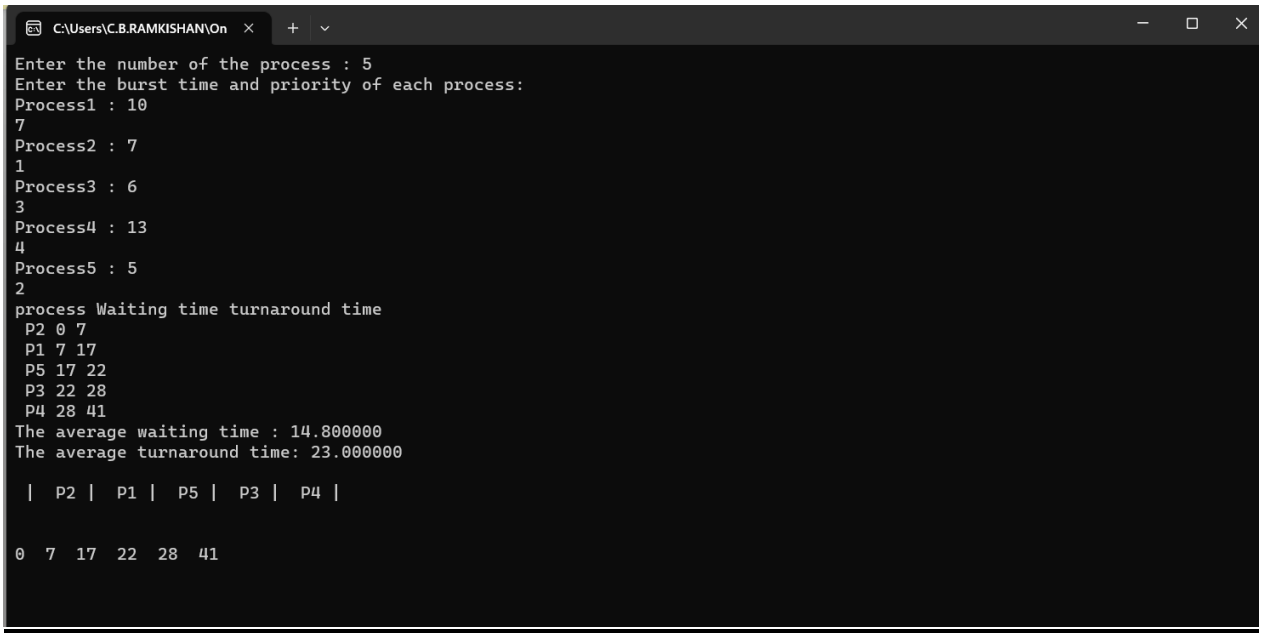
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int n,process[10],bt[10],priority[10],wt[10],tat[10];
void sortbt()
{
        int i,j,pos,temp;
        for(i=0;i<n;i++)
        {
                pos=i;
                for(j=i+1;j<n;j++)
                {
                        if(priority[j]<priority[pos])
                                pos=j;
                }
                temp=bt[pos];
                bt[pos]=bt[i];
                bt[i]=temp;
                temp=process[pos];
                process[pos]=process[i];
                process[i]=temp;
        }
}
```

```
void waitingtime()
{
        int i;
        wt[0]=00;
        for(i=1;i<n;i++)
wt[i]=bt[i-1]+wt[i-1];
}
void turnaroundtime()
{
        int i;
        for(i=0;i<n;i++)
                tat[i]=bt[i]+wt[i];
}
void average()
{
        float awt,atat;
        int twt=0,ttat=0,i;
        sortbt();
        waitingtime();
        turnaroundtime();
        printf("process Waiting time turnaround time\n");
        for(i=0;i<n;i++)
        {
                printf(" P%d %d %d\n",process[i],wt[i],tat[i]);
                twt+=wt[i];
                ttat+=tat[i];
        }
        awt=(float)twt/n;
        atat=(float)ttat/n;
        printf("The average waiting time : %f\n",awt);
        printf("The average turnaround time: %f\n",atat);
}
void ganttchart()
{
        int i;
        printf(" \n");
        printf(" | ");
        for(i=0;i<n;i++)
        printf(" P%d | ",process[i]);
        printf("\n \n");
        printf("\n0 ");
        for(i=0;i<n;i++)
        printf(" %d ",tat[i]);
}
int main()
{
        int i;
        printf("Enter the number of the process : ");
        scanf("%d",&n);
        printf("Enter the burst time and priority of each process: \n");
        for(i=0;i<n;i++)
```

```
    {
        process[i]=i+1;
        printf("Process%d : ",process[i]);
        scanf("%d",&bt[i]);

        scanf("%d",&priority[i]);
    }
    average();
    ganttchart();
    getch();
    return 0;
}
```

OUTPUT:



Observation	
Record	
Total	
Initial	

RESULT:

Thus waiting time & turnaround time for processes based on Priority scheduling was computed and the average waiting time was determined.

Ex No:2.b **Round Robin**

Date : 08.02.2023

AIM:

To schedule snapshot of processes queued according to Round robin scheduling.

ALGORITHM:

1. Get length of the ready queue, i.e., number of process (say n)
2. Obtain Burst time Bi for each processes Pi.
3. Get the time slice per round, say TS
4. Determine the number of rounds for each process.
5. The wait time for first process is 0.
6. If Bi > TS then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round.
7. Calculate average waiting time and turn around time
8. Display the GANTT chart that includes

a. order in which the processes were processed in progression of rounds

b. Turnaround time Ti for each process in progression of rounds.

9. Display the burst time, turnaround time and wait time for each process (in order of rounds they were processed).
10. Display average wait time and turnaround time
11. Stop

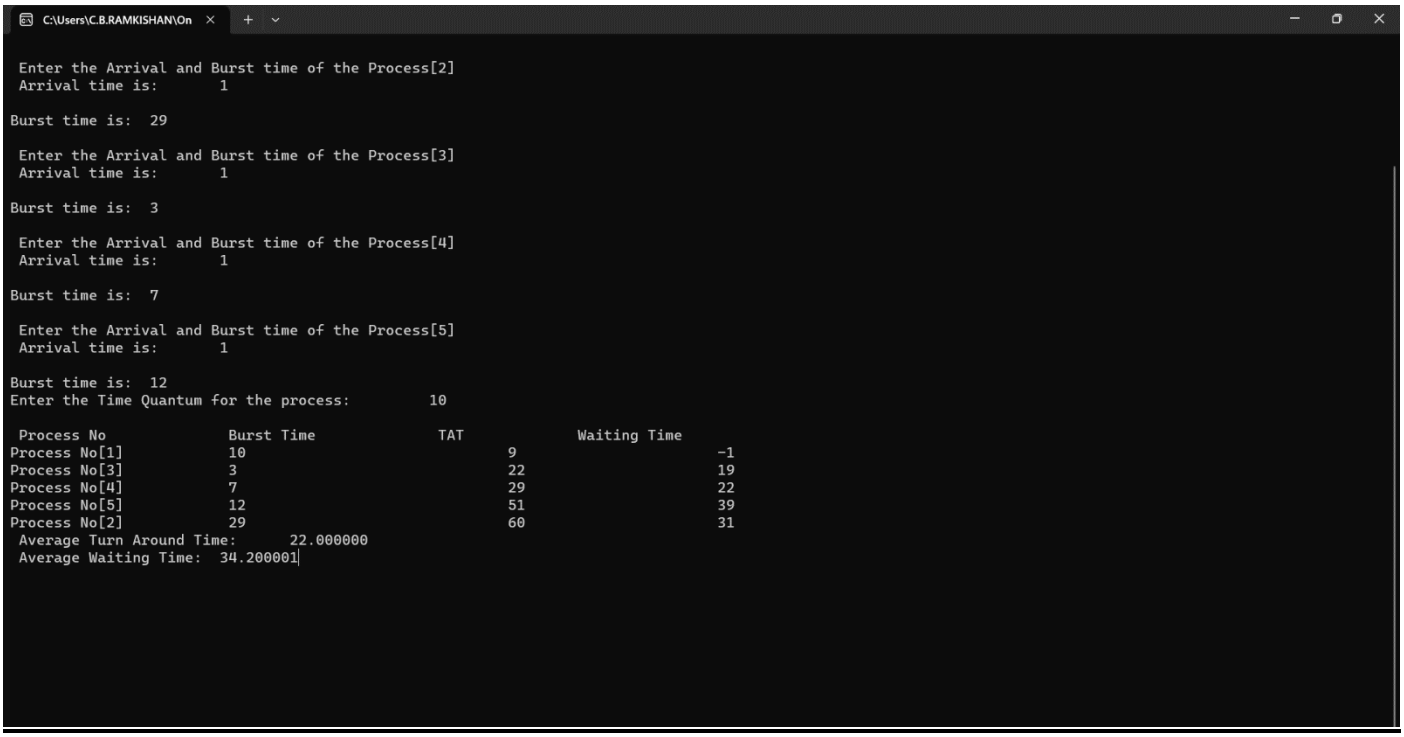
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
        printf(" Total number of process in the system: ");
        scanf("%d", &NOP);
```

```
y = NOP;
for(i=0; i<NOP; i++)
{
   printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf(" Arrival time is: \t");
        scanf("%d", &at[i]);
        printf(" \nBurst time is: \t");
        scanf("%d", &bt[i]);
        temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
        if(temp[i] <= quant && temp[i] > 0)
        {
                sum = sum + temp[i];
                temp[i] = 0;
                count=1;
        }
        else if(temp[i] > 0)
        {
                temp[i] = temp[i] - quant;
                sum = sum + quant;
        }
```

```
    if(temp[i]==0 && count==1)
                {
                        y--;
                printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
                        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
                        count =0;
                }
                if(i==NOP-1)
                {
                        i=0;
                }
                else if(at[i+1]<=sum)
                {
                        i++;
                }
                else
                {
                        i=0;
                }
        }
        avg_wt = wt * 1.0/NOP;
        avg_tat = tat * 1.0/NOP;
        printf("\n Average Turn Around Time: \t%f", avg_wt);
        printf("\n Average Waiting Time: \t%f", avg_tat);
        getch();
}
```

OUTPUT:



Observation	
Record	
Total	
Initial	

RESULT:

Thus waiting time & turnaround time for processes based on Round Robin was computed and the average waiting time was determined.

Ex.No:3 IMPLEMENTATION OF DEADLOCK AVOIDANCE ALGORITHM

Date :17.02.2023

AIM:

To write a 'C' program by avoiding the deadlock using banker's algorithm.

ALGORITHM:

- Get the allocated and maximum resources and process name from the user.
- Calculate the available matrix from the maximum and allocated resources.
- Then calculate the need matrix.
- Then apply the banker's algorithm to allocate the requested resource from available resources.
- Release the allocated resource from completed process.
- This step is repeated to find the safe state among the available process.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
void main()
 {
int
k=0,a=0,b=0,instance[5],availability[5],op[10],allocated[10][5],need[10][5],MAX[10][5],process,P[10],no_of_r
esources, cnt=0,i, j;
printf("\n Enter the number of resources : ");
scanf("%d", &no_of_resources);
printf("\n enter the max instances of each resources\n");
for (i=0;i<no_of_resources;i++) {
availability[i]=0;
printf("%c= ",(i+97));
scanf("%d",&instance[i]);
}
printf("\n Enter the number of processes : ");
scanf("%d", &process);
printf("\n Enter the allocation matrix \n     ");
for (i=0;i<no_of_resources;i++)
printf(" %c",(i+97));
printf("\n");
for (i=0;i <process;i++) {
P[i]=i;
printf("P[%d]  ",P[i]);
```

```
for (j=0;j<no_of_resources;j++) {



scanf("%d",&allocated[i][j]);
availability[j]+=allocated[i][j];
}
}
printf("\nEnter the MAX matrix \n     ");
for (i=0;i<no_of_resources;i++) {
printf(" %c",(i+97));
availability[i]=instance[i]-availability[i];
}
printf("\n");
for (i=0;i <process;i++) {
printf("P[%d]  ",i);
for (j=0;j<no_of_resources;j++)
scanf("%d", &MAX[i][j]);
}
printf("\n");
A: a=-1;
for (i=0;i <process;i++) {
cnt=0;
b=P[i];
for (j=0;j<no_of_resources;j++) {
need[b][j] = MAX[b][j]-allocated[b][j];
if(need[b][j]<=availability[j])
cnt++;
}
if(cnt==no_of_resources) {
op[k++]=P[i];
for (j=0;j<no_of_resources;j++)
availability[j]+=allocated[b][j];
} else
P[++a]=P[i];
}
if(a!=-1) {
process=a+1;
goto A;
}
printf("\t <");
for (i=0;i<k;i++)
printf(" P[%d] ",op[i]);
printf(">");
return 0;
}
```

OUTPUT:

Observation	
Record	
Total	
Initial	

RESULT:

Thus the implementation of deadlock avoidance using bankers algorithm was determined.