Ex. No.: 6a)
Date: 20.2.25

FIRST COME FIRST SERVE

Aim:

To implement First-come First- serve (FCFS) scheduling technique

Algorithm:

1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

Program Code:

```
# include <stdio.h>
int main () {
int num;
printf (" Enter the number of process:");
scanf(" %.d", &num);
int bt [n];
printf ("Burst Time:");
for (int i = 0 ; i<n ; i++) {
        scanf ("%.d", & bt[i]);{
int ct[n];
printf (" Completion time:");
```

```
int count=0;
for (int i=0 ; i<n; i++) {
    count += bt[i];
    count+[i] = count ;
    printf ("%d\n", ct[i]);}
    int tt,[n], int [n];
    printf ("Turn around time : \n");
    for (int i=0; i<n; i++){
        ++[i] = ct[i];
        printf ("%. d\n", tt[i]); }
    printf (" Waiting time : \n");
    for (int i=0 ; i<n ; i++ ){
        wt[i] = tt[i] - bt [i];
        printf ("%.d\n", wt[i]);}
int avg - wt =0. avg -tt=0;
for (int i=0 ; i<n; i++) {
    avg-wt t = wt[i];
    avg-ttt = tt[i];
    avg - wt = avg- wt /n;
    avg-tt = avg-tt/n;
    printf (" Average waiting time : %.d\n",
                                avg_wt");
    printf (" Average turn around time : %.d\n",
                                avg-tt);
}
```

Sample Output:

Enter the number of process:

3

Enter the burst time of the processes:

24 3 3

Process	Burst Time	Waiting Time	Turn Around Time
0	24	0	24
1	3	24	27
2	3	27	30

Average waiting time is: 17.0

Average Turn around Time is: 19.0

Output:

Enter the number of process: 3

Enter the burst time of the processes:

24 3 3

Process	Burst Time	Waiting Time	Turn Around Time
0	24	0	24
1	3	24	27
2	3	27	30

Average waiting Time is : 17.0 ms

Average Turn around Time is : 19.0 ms.

Result:

Hence the fcfs (First come first serve) scheduling is verified.

Ex. No.: 6b)
Date:

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero. 5. Sort based on burst time of all processes in ascending order 6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time. 8. Display the results.

Program Code:

```
#include<stdio.h>
int main() {
int n;
printf("Enter No:of processes:");
Scanf("%d", &n);
int P[n], bt[n], wt[n], tat[n];
Float total_wt =0, total_tat=0;
printf("Enter burst time for the each processes:\n");
for(int i=0; i<n ; i++){
        Scanf ("%d", &bt[i]);
}
for (int i=0; j<n; i++) { for(int j = i ; j<n; j++) {
        int temp= bt[i];          if (bt[i] > bt[j]) {
        bt[i] = bt[j];
        bt[j] = temp;
        temp = p[i];
        P[i] = P[j];
```

```
            P[j] = temp;
        }

    }
}
wt [0] = 0
For (int i=1 ; i<n; i++) {
        wt [i] = bt [i-1] + wt [i-1]
}
for (int i = 0 ; i<n; i++) {
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
}
printf ("Process \t Burst time \t Waiting
        time \t TAT \n")
for (int i = 0; i<n; i++) {
    printf ("%d \t %d \t %d \t
        %d \t \n", P[i], bt[i], wt[i], tat[i])
}
float avg_wt, avg_tat;
avg_wt = total_wt/n;
avg_tat = total_tat /n;
printf ("Avg waiting time: %f ", avg_wt);
```

```
    printf("Avg tat : %.1B", avg_tat);
}
```

Input

Enter the no: of process: 4

Enter the burst time for all process :

6 8 7 3

Burst Time

3

6

7

8

Process	Burst Time (ms)	Waiting time (ms)	turn around time (ms)
0	3	0	3
1	6	3	9
2	7	9	16
3	8	16	24

Sample Output:

Enter the number of process:
4
Enter the burst time of the processes:
8 4 9 5

Process	Burst Time	Waiting Time	Turn Around Time
2	4	0	4
4	5	4	9
1	8	9	17
3	9	17	26

Average waiting time is: 7.5
Average Turn Around Time is: 13.0

Result:

Thus the shortest Job First algorithm is executed.

Ex. No.: 6c)
Date:

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority 4.
Calculate the total waiting time and total turnaround time for each process 5.
Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time

Program Code:

```
#include <stdio.h>
# include <stdlib.h>
int main() {
    int n;
    printf ("Enter no of process: \n");
    scanf ("%d", &n);
    int bt [n], p[n], ct[n], wt =0, tat[n], wb[n],
at at = 0, cwt = 0;
printf ("Enter the burst time");
for (int i=0; i<n; i++)
    scanf ("%d", & bt [i]);
printf ("Enter the priority of processes: \n");
for(int i=0; i<n41, i++)
    scanf ("%d", &p [i]);
int sp [n];
```

```
for (int i=0 ; i<n; i++)
    printf ("%.d\n", wt [i]);
for (int  i=0 ; i<n ; i++){
    atat  = atat + tat [i];
    awt = awt  twt [i] ; }
printf ("\n Average Turnaround Time : %.2f\n
  Average wait time: %.2f \n", (f wat) atat /n,
        (float) a wt /n);
```

output:

Enter the no: of process : 4

Enter the burst time : 1 3
5
8
4

Enter the priority of process:
3
4
2
1

Completion time : 22
9
30
4

Turn around Time : 22
9
30
4

```
for (int i=0 ; i<n; i++)
    sp[i] = p[i];
for (int i=0 ; i<n-1; i++) {
    for (int j=0 ; j<n-1 ; j++){
    if (sp [j+1] <sp [j]){
int temp =sp [j+1];
sp [j+1] =sp [j];
sp [i] =temp ; }}}
int c=0;
For (int i=0; i<n; i++) {
    for (int j=0 ;j<n ; j++) {
        if (sp[i] ==p [j]) {
            ct [j] = c+bt [j];
              c = ct [j]
             tat [j] = ct [j] - at;
              wt [j] =tat [j] - bt [j]; } }}
printf ("\n completion Time \n");
for (int i=0; i<n; i++)
    printf ("%.d\n", ct[i]);
printf ("\n Turn around time \n");
 for (int i=0; i<n;i++)
        printf ("%.d \n", tat[i]);
printf ("\n wait time\n");
```

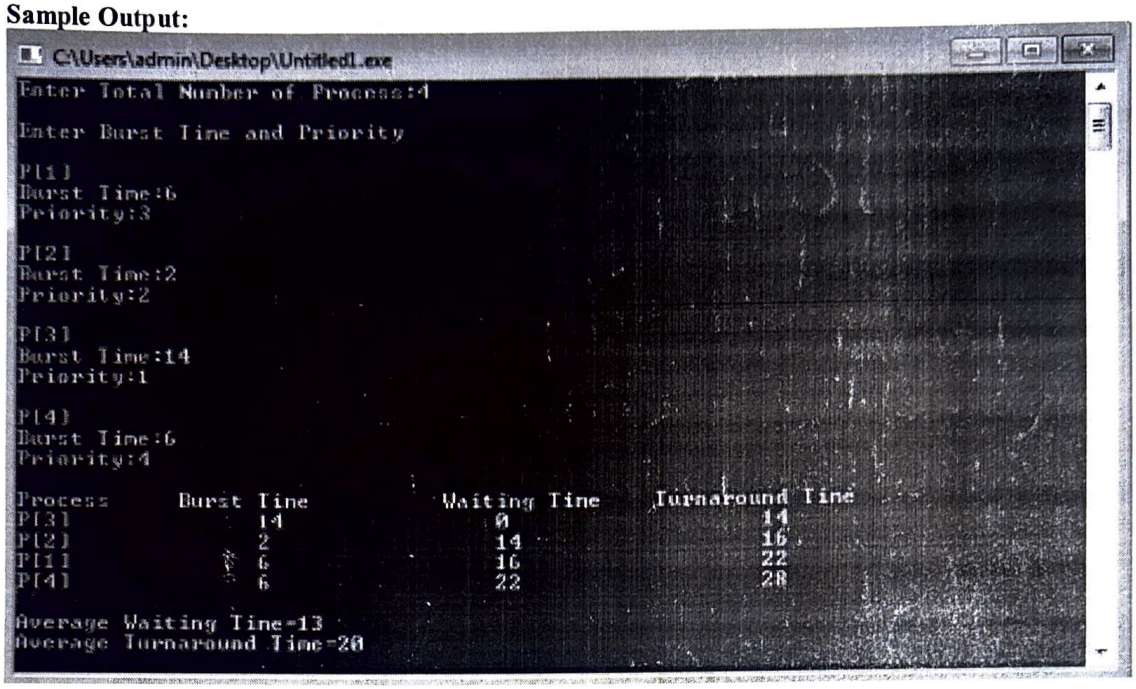
Process	BT (ms)	Priority	CT (ms)	TAT (ms)	WT
P_1	13	3	22	22	9
P_2	5	2	9	9	4
P_3	8	4	30	30	22
P_4	4	1	4	4	0

Average Turnaround Time: 16.25 ms

Average wait time: 8.25 ms

P_4 P_2 P_1 P_3

Sample Output:



wait time : 9

4

22

0

Average Turnaround Time : 16.25 ms

Average wait time : 8.75 ms

Result:

The priority scheduling technique is impleted using C.

Ex. No.: 6d)
Date

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0. 6. Initialize time : t = 0
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
   a- If rem_bt[i] > quantum
   (i) t = t + quantum
   (ii) bt_rem[i] -= quantum;
   b- Else // Last cycle for this process
   (i) t = t + bt_rem[i];
   (ii) wt[i] = t - bt[i]
   (iii) bt_rem[i] = 0; // This process is over
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

Program Code:

```
#include<stdio.h>
void RR (int bt[], int at[], int n, int q)
{
    int ct[n], tat[n], wt[n];
    int rem_bt[n];
    int to_tat=0; tat_cot=0;
    for (int i=0; i<n; i++)
        rem_bt[i] = bt[i];
int t=0;
int pr_left=n;
while (pr_left>0)
{
    int done_this_round=0;
```

```
for (int i=0; i<n; i++)
  {
      if (rem_bt[i]>0 && at[i]<=t)
          { done_this_round=1;

if (rem_bt[i]>q)
 {
    t+=q;
    rem_bt[i]-=q;
  }
 else{
    t+=rem_bt[i];
    ct[i]=t;
    rem_bt[i]=0;
    pr_left--;
  }
  }
}
if (!done_this_round)
    t++;
}
for (int i=0; i<n; i++)
{ tat[i]=ct[i]-at[i];
  wt[i]=tat[i]-bt[i];
  tat_tat+=tat[i];
  tat_wt+=wt[i];
}
```

```
printf ("\n Process \t BT \t AT \t CT \t TAT \t wt\n");
for (int i=0 ; i<n ; i++)
    printf (i+1, bt[i], at[i], ct[i], tat[i], wt[i]);
printf ((float) tot_tat /n, (float) tot_wt /n);
}
int main()
{
  int n, q;
  scanf ("%.d", &n);
  int bt[n], at[n];
  for (int i=0; i<n; i++)
  {
     scanf ("%.d", & bt[i]);
  }
for (int i=0 ; i<n; i++)
  {
     scanf ("%.d", & at[i]);
  }
scanf ("%.d", &q);
   (bt, at, n, q);
 return 0;
}
```

OUTPUT:

Total no: of process : 3

Details of process : 1

Arrival Time : 0

Burst Time : 4

Enter Details of process : 2

Arrival Time : 1

Burst Time : 7

Details of process : 3

Arrival Time : 2

Burst Time : 5

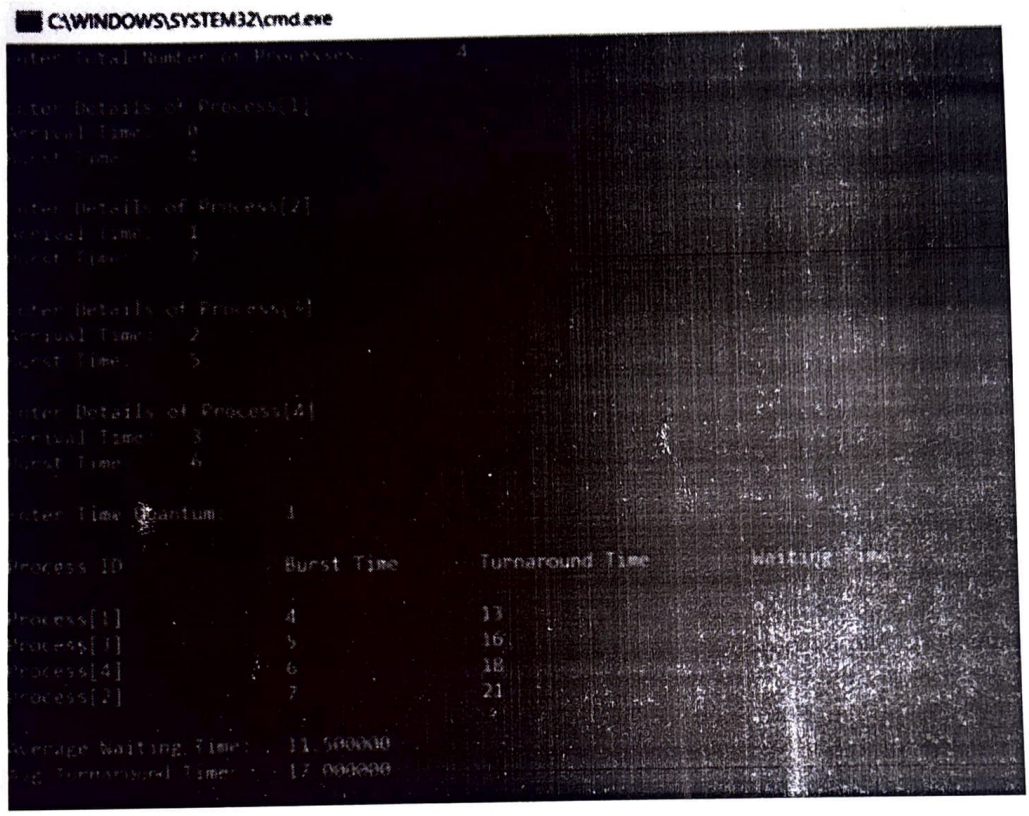
quantum : 2

process ID	BT	TAT	WT
1	4	8	4
3	5	13	8
2	7	15	8

Average waiting time : 6.66 ms

Average Turn around time : 12.00 ms

Sample Output:



Result:

Thus the given code for Round Robin algorithm is executed successfully.