Ex. No.: 6a)
Date: 21/2/25

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 n;
    printf("Enter the number of process:");
    scanf("%d",&n);
    int burst[n];
    printf("Enter the burst time of the processes:");
    for (int i=0; i<n; i++)
        scanf("%d",&burst[i]);
    printf("process    burst time    waiting time    turn around
    time\n");
    int wt=0, tat=burst[0];
    float avg_wt=0, avg_tat=0;
```

```
for (int i=0;i<n; i++)
{
    printf("%d        %d        %d        %d\n", i,
    burst[i], wt, tat);
    avg_wt += wt;
    avg_tat += tat;
    wt = wt + burst[i];
    tat = burst[i+1] + wt;
}
avg_wt = avg_wt/n;
avg_tat = avg_tat/n;
printf("Average waiting time is: %.1f\n", avg_wt);
printf("Average Turn Around time is: %.1f", avg_tat);
}
```

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

Result:

Program to implement First Come First Serve Scheduling Technique was executed Successfully.

Output:

Enter number of process : 3

Enter the burst time of the processes: 5 3 8

process	Burst time	Waiting Time	Turn Around Time
0	5	0	5
1	3	5	8
2	8	8	16

Average waiting Time is : 4.3

Average Turn Around Time is : 9.7

Gantt Chart:

Process	Arrival Time	Burst time	Completion Time	waiting time TAT - BT	Turn Around Time CT - AT
P_1	0	5	5	0	5
P_2	0	3	8	5	8
P_3	0	8	16	8	16

Average waiting time : $\frac{0+5+8}{3}$ = 4.3

Average Turn Around time : $\frac{5+8+16}{3}$: 9.7

Ex. No.: 6b)
Date: 22/2/25

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 number of processes: ");
    scanf("%d", &n);
    int burst[n];
    for (int i=0; i<n; i++){
        scanf("%d", &burst[i]);
    }
    int wt=0, max, tat=0, min, ind=0;
    float avg_wt=0, avg_tat=0;
```

```
for (int i=0; i<n;i++){
    if (burst[i] > max){
        max = burst[i];
    }
}
max = max+1;
for (int i=0 ; i<n;i++){
    min=max;
    for (int j=0 ; j<n ;j++){
        if (burst[j] < min){
            min = burst[j];
            ind = j;
        }
    }
    tat = min + wt;
    printf ("%d %d %d %d\n",ind+1, min , wt , tat);
    avg_wt += wt;
    wt += min;
    avg_tat += tat;
    burst[ind] = max;
}
printf("Average waiting time is: %.1f\n",(avg_wt/n));
printf("Average turn around time is: %.f\n", (avg_tat/n));
return 0;
}
```

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:

Program to implement Shortest Job First Scheduling Algorithm was written & executed successfully.

Output:

Enter the number of process: 4

Enter the burst time of the process: 8 4 6 3

Process	Burst Time	Waiting Time	Turn Around Time
4	3	0	3
2	4	3	7
3	6	7	13
1	8	13	21

Average waiting time is : 5.75 ms

Average TurnAround Time is : 11 ms

Gantt Chart:

Process	Arrival time	Burst time	Completion time	Waiting time WT = TAT - BT	Turn around time TAT = CT - AT
1	0	8	21	13	21
2	0	4	7	3	7
3	0	6	13	7	13
4	0	3	3	0	3

Average waiting time = $\frac{13+3+7+0}{4} = \frac{23}{4} = 5.75$ ms

Average Turn Around time is = $\frac{21+7+13+3}{4} = \frac{44}{4} = 11$ ms

Ex. No.: 6c)
Date: 8/3/25

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

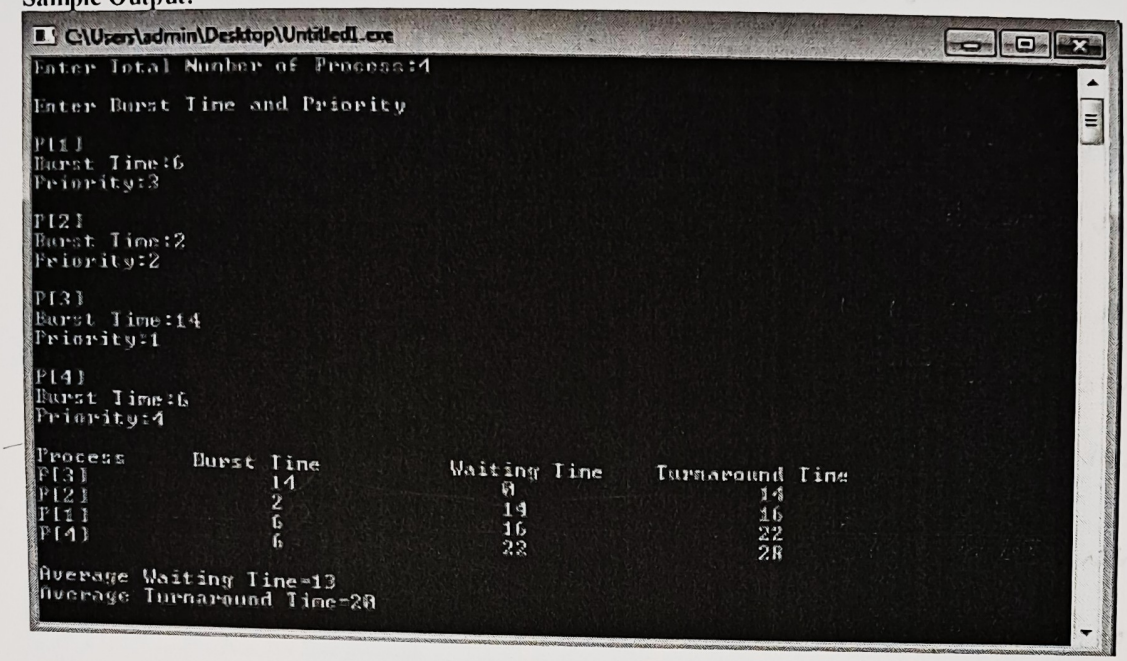
1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority
4. Calculate the total waiting time and total turnaround time for each process
5. Display the process name & burst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

Program Code:

```
#include <stdio.h>
int main(){
    int n, temp;
    printf("Enter the number of process:");
    scanf("%d", &n);
    int p[3][n];
    for (int i=0; i<n; i++)
    {
        p[0][i] = i+1;
        printf("P[%d]\nBurst Time:", i+1);
        scanf("%d", &p[1][i]);
        printf("Priority:");
        scanf("%d", &p[2][i]);
    }
    for (int i= 0; i<n-1; i++){
        for (int j=0; j<n-1-i; j++){
```

```
        if ( p[2][j] > p[2][j+1])
        {
            for (int k=0 ; k<3; k++)
            {
                temp = p[k][j];
                p[k][j] = p[k][j+1];
                p[k][j+1] = temp;
            }
        }
    }
}
int wt=0 , tat = p[i][0];
float avg_wt = 0,avg_tat = 0;
for (int j=0 ; j<n ; j++)
{
    printf ("p[%d]  %d  %d  %d", p[0][j], p[1][j],
                                 wt ,tat);
    avg_wt += wt;
    avg_tat += tat;
    wt += p[1][j];
    tat += p[i][j+1];
}
printf ("Average Waiting Time: %.1f", avg_wt/n);
printf ("Average Turnaround Time: %.1f", avg_tat/n);
}
```

Sample Output:



Result:

Program to implement a priority scheduling algorithm was written and executed succesfully.

Output:

Enter Total Number of Process: 3

Enter burst time and Priority:

P[1]

Burst Time: 6

Priority: 3

P[2]

Burst Time: 2

Priority: 1

P[3]

Burst time 6

Priority: 2

Process	Burst Time	Waiting Time	Turn Around Time
P[2]	2	0	2
P[3]	6	2	8
P[1]	6	8	14

Gantt chart

Process	Burst time BT(ms)	Arrival time AT (ms)	Completion time CT (ms)	Turn Around time TAT = CT - AT (ms)	Waiting time WT = TAT - BT
1	6	0	14	14	8
2	2	0	2	2	0
3	6	0	8	8	2

Ex. No.: 6d)
Date 21/3/25

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.

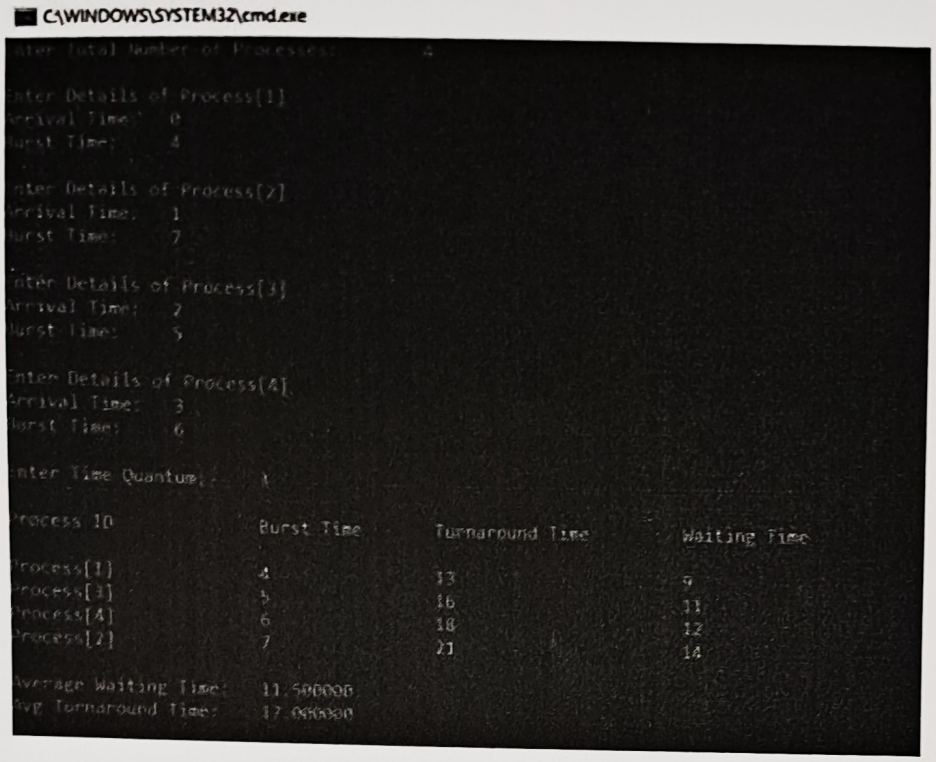
Program Code:

```
#include <stdio.h>
struct process {
    int pid;
    int at;
    int bt;
    int ct;
    int rem;
    int wt;
    int tat;
};
```

```
void sort (struct process p[], int n)
{
    for (int i=0; i<n-1; i++) {
        for (int j=0; j<n-i-1; j++) {
            if (p[j].at > p[j+1].at) {
                struct process t = p[j];
                p[j] = p[j+1];
                p[j+1] = t;
            }
        }
    }
}
int main() {
    int n;
    printf ("Enter the number of processes: ");
    scanf ("%d", &n);
    struct process p[n];
    for (int i=0; i<n; i++)
    {   printf ("Enter Details of process [%d] \n", i+1);
        p[i].pid = i+1;
        printf ("Arrival Time");
        scanf ("%d", &p[i].at);
        printf ("Burst Time:");
        scanf ("%d", &p[i].bt);
        p[i].rem = p[i].bt;
        p[i].ct = 0
    }
```

```
int q;
printf("Enter time Quantum:");
scanf("%d", &q);

int t = 0, c = 0;

while (c<n) {

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

        if (p[i].rem > q)
        {
            t+ = q;
            p[i].rem -= q;
        }
        else if (p[i].ct == 0)
        {
            t+ = p[i].rem;
            p[i].rem = 0;
            p[i].ct = t;
            p[i].tat = p[i].ct - p[i].at;
            p[i].wt = p[i].tat - p[i].bt;
            c++;
        }
    }
}
printf("ProcessID  Burst Time  Turn Around Time  Waiting Time\n");

float twt=0, ttat=0;

for (int i=0; i<n; i++){
    twt = twt + p[i].wt;
    ttat = ttat + p[i].tat;
    printf("%d  %d  %d  %d\n", p[i].pid,
                p[i].bt, p[i].tat, p[i].wt);
}
printf("Average Waiting Time = %.f", (twt/n));
printf("\nAverage Turn Around Time = %.f", (ttat/n));
}
```

Sample Output:



Result:

Program to implement a Round Robin Scheduling Algorithm was written & executed successfully.

Output:

Enter number of processes : 4

Enter Details of process[1]:

Arrival Time : 2

Burst Time : 5

Enter Details of process[2]:

Arrival Time: 1

Burst Time : 4

Enter Details of process[3]:

Arrival Time : 3

Burst Time : 1

Enter Details of process[4]:

Arrival Time : 4

Burst Time : 4

Enter time Quantum : 3

Process ID	Burst Time	Turn Around Time	Waiting Time
1	5	10	5
2	4	12	8
3	1	4	3
4	4	10	6

Average Waiting Time = 5.5

Average Turn Around Time = 9.0

Grantt Chart :

Process	Burst time	Arrival time	Completion Time	Turn Around Time	Waiting time
1	5	2	12	10	5
2	4	1	13	12	8
3	1	3	7	4	3
4	4	4	14	10	6

Average waiting time = $\frac{5+8+3+6}{4} = \frac{22}{4} = 5.5$ ms

Average Turn Around time : $\frac{10+12+4+10}{4} = \frac{36}{4} = 9.0$ ms