NAME:PRIYANKA E ROLL NO:231901038

Exp:6a

Date:22.2.25

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 pno;
    printf("Enter the number of processes: ");
    scanf("%d", &pno);

    int bt[pno], p[pno], wt[pno], tat[pno];
    float avg_wt = 0, avg_tat = 0;

    printf("Enter the burst time of the processes: ");
    for (int i = 0; i < pno; i++) {
        scanf("%d", &bt[i]);
        p[i] = i;
    }

    wt[0] = 0;
    for (int i = 1; i < pno; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
    }
```

```
    for (int i = 0; i < pno; i++) {
        tat[i] = wt[i] + bt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= pno;
    avg_tat /= pno;

    printf("Process\tBurst Time\tWaiting Time\tTurn Around Time\n");
    for (int i = 0; i < pno; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", avg_wt);
    printf("Average Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

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

```
┌──(student㉿kali)-[~]
└─$ vi fcfs.c

┌──(student㉿kali)-[~]
└─$ gcc fcfs.c -o fcfs

┌──(student㉿kali)-[~]
└─$ ./fcfs
Enter the number of processes: 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: 17.00
Average Turnaround Time is: 27.00
```

Result:

Thus,the fcfs has been successfully executed.

Ex. No.: 6b)

Date: 1.3.25

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:

Non preemptive :

```
#include <stdio.h>

int main() {

   int n, i, j, temp;

   float total_wt = 0, total_tat = 0;



   printf("Enter the number of processes: ");
```

```
    scanf("%d", &n);



    int at[n], bt[n], pid[n], wt[n], tat[n];



    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (i = 0; i < n; i++) {
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &at[i]);
        printf("Burst Time: ");
        scanf("%d", &bt[i]);
        pid[i] = i + 1;
    }
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if ((at[j] > at[j + 1]) || (at[j] == at[j + 1] && bt[j] > bt[j +
1])) {
```

```
            temp = at[j];
            at[j] = at[j + 1];
            at[j + 1] = temp;
            temp = bt[j];
            bt[j] = bt[j + 1];
            bt[j + 1] = temp;


            temp = pid[j];
            pid[j] = pid[j + 1];
            pid[j + 1] = temp;
        }
    }
}
int completion_time = 0;
for (i = 0; i < n; i++) {
    if (completion_time < at[i])
```

```
            completion_time = at[i];

        wt[i] = completion_time - at[i];

        tat[i] = wt[i] + bt[i];

        completion_time += bt[i];

        total_wt += wt[i];

        total_tat += tat[i];

    }

    printf("\nNon-Preemptive SJF Scheduling\n");

    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurn
Around Time\n");

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

        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i], wt[i],
tat[i]);

    }

    printf("\nAverage Waiting Time: %.2f", total_wt / n);

    printf("\nAverage Turn Around Time: %.2f\n", total_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
```

Output:

```
┌──(student㉿kali)-[~]
└─$ vi sjfnon.c

┌──(student㉿kali)-[~]
└─$ gcc sjfnon.c -o sjfnon

┌──(student㉿kali)-[~]
└─$ ./sjfnon
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 0
Burst Time: 8
Process 2:
Arrival Time: 0
Burst Time: 4
Process 3:
Arrival Time: 0
Burst Time: 9
Process 4:
Arrival Time: 0
Burst Time: 5

Non-Preemptive SJF Scheduling

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

Average Waiting Time: 7.50
Average Turn Around Time: 14.00
```

Preemptive:

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

int main() {
    int n, i, smallest, time, remain, endTime;
    float total_wt = 0, total_tat = 0;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    int at[n], bt[n], rt[n], pid[n], wt[n], tat[n];
```

```
printf("Enter Arrival Time and Burst Time for each process:\n");
for (i = 0; i < n; i++) {
   printf("Process %d:\n", i + 1);
   printf("Arrival Time: ");
   scanf("%d", &at[i]);
   printf("Burst Time: ");
   scanf("%d", &bt[i]);
   rt[i] = bt[i];
   pid[i] = i + 1;
}

remain = 0;
time = 0;

printf("\nPreemptive SJF (SRTF) Execution Order:\n");

while (remain < n) {
   smallest = -1;
   int min_bt = INT_MAX;

   for (i = 0; i < n; i++) {
      if (at[i] <= time && rt[i] > 0 && rt[i] < min_bt) {
         min_bt = rt[i];
         smallest = i;
      }
   }

   if (smallest == -1) {
      time++;
      continue;
   }

   rt[smallest]--;
   time++;

   if (rt[smallest] == 0) {
      remain++;
      endTime = time;
      tat[smallest] = endTime - at[smallest];
      wt[smallest] = tat[smallest] - bt[smallest];
```

```
            total_wt += wt[smallest];
            total_tat += tat[smallest];
        }
    }

    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurn Around Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turn Around Time: %.2f\n", total_tat / n);

    return 0;
}
```

Output:

```
┌──(student㉿kali)-[~]
└─$ vi sjfpre.c

┌──(student㉿kali)-[~]
└─$ gcc sjfpre.c -o sjfpre

┌──(student㉿kali)-[~]
└─$ ./sjfpre
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 0
Burst Time: 8
Process 2:
Arrival Time: 0
Burst Time: 4
Process 3:
Arrival Time: 0
Burst Time: 9
Process 4:
Arrival Time: 0
Burst Time: 5

Preemptive SJF (SRTF) Execution Order:

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

Average Waiting Time: 7.50
Average Turn Around Time: 14.00
```

Result:

Thus,the sjf has been successfully executed.

Ex. No.: 6c)
Date: 21.3.25

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:

Non preemptive

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter total number of processes: ");
    scanf("%d", &n);

    int id[n], burst_time[n], priority[n], waiting_time[n], turnaround_time[n];

    for (int i = 0; i < n; i++) {
        id[i] = i + 1;
        printf("Enter Burst Time and Priority for P%d: ", id[i]);
        scanf("%d %d", &burst_time[i], &priority[i]);
    }

    // Sorting based on priority (lower value = higher priority)
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (priority[i] > priority[j]) {
```

```
            int temp;
            temp = priority[i]; priority[i] = priority[j]; priority[j] = temp;
            temp = burst_time[i]; burst_time[i] = burst_time[j]; burst_time[j] = temp;
            temp = id[i]; id[i] = id[j]; id[j] = temp;
        }
    }
}

waiting_time[0] = 0;
turnaround_time[0] = burst_time[0];

for (int i = 1; i < n; i++) {
    waiting_time[i] = waiting_time[i - 1] + burst_time[i - 1];
    turnaround_time[i] = waiting_time[i] + burst_time[i];
}

// Display Results
printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
float total_wt = 0, total_tat = 0;
for (int i = 0; i < n; i++) {
    printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", id[i], burst_time[i], priority[i], waiting_time[i],
turnaround_time[i]);
    total_wt += waiting_time[i];
    total_tat += turnaround_time[i];
}

printf("\nAverage Waiting Time: %.2f", total_wt / n);
printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

return 0;
}
```

Output:

```
┌──(student㉿kali)-[~]
└─$ vi prinon.c

┌──(student㉿kali)-[~]
└─$ gcc prinon.c -o prinon

┌──(student㉿kali)-[~]
└─$ ./prinon
Enter total number of processes: 4
Enter Burst Time and Priority for P1:
6
3
Enter Burst Time and Priority for P2: 2
2
Enter Burst Time and Priority for P3: 14
1
Enter Burst Time and Priority for P4: 6
4

Process Burst Time      Priority        Waiting Time    Turnaround Time
P3      14              1               0               14
P2      2               2               14              16
P1      6               3               16              22
P4      6               4               22              28

Average Waiting Time: 13.00
Average Turnaround Time: 20.00
```

Preemptive

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter total number of processes: ");
    scanf("%d", &n);

    int id[n], burst_time[n], priority[n], arrival_time[n], waiting_time[n], turnaround_time[n],
remaining_time[n];

    for (int i = 0; i < n; i++) {
        id[i] = i + 1;
        printf("Enter Arrival Time, Burst Time, and Priority for P%d: ", id[i]);
        scanf("%d %d %d", &arrival_time[i], &burst_time[i], &priority[i]);
        remaining_time[i] = burst_time[i];
    }
```

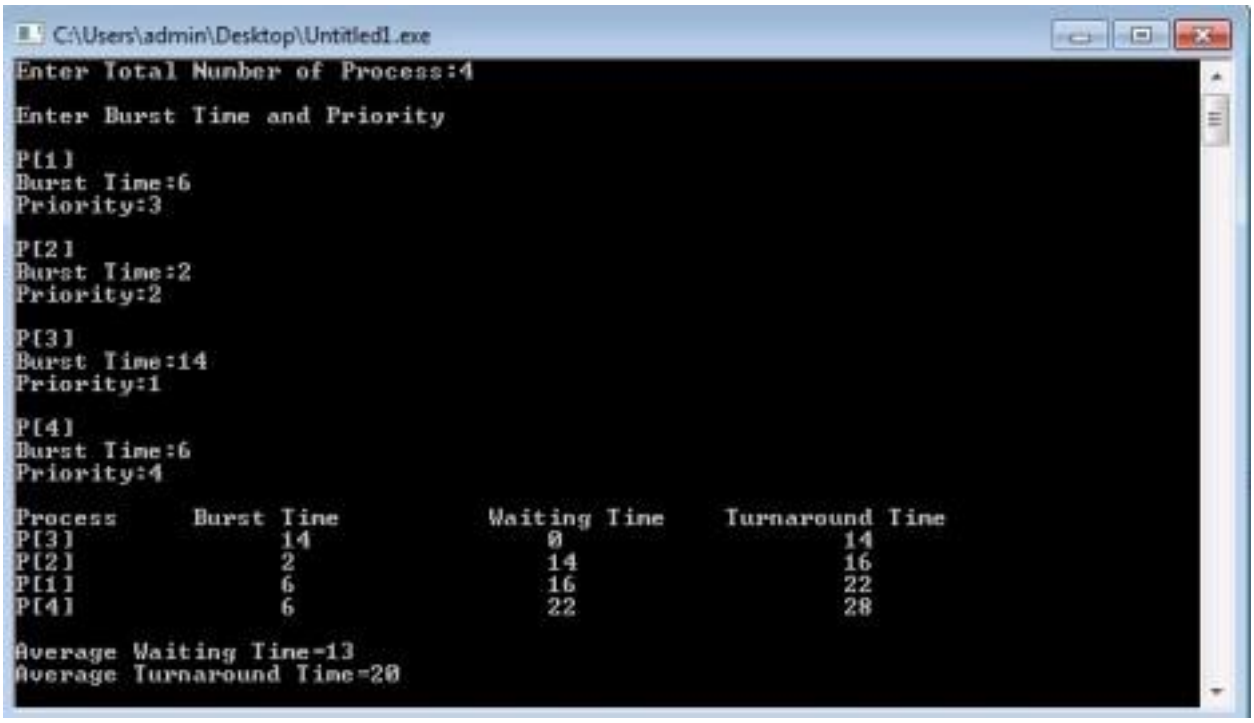
```
    int completed = 0, time = 0;
    while (completed < n) {
        int min_priority = 9999, min_index = -1;
        for (int i = 0; i < n; i++) {
            if (arrival_time[i] <= time && remaining_time[i] > 0 && priority[i] < min_priority) {
                min_priority = priority[i];
                min_index = i;
            }
        }

        if (min_index == -1) {
            time++;
            continue;
        }

        remaining_time[min_index]--;
        time++;

        if (remaining_time[min_index] == 0) {
            completed++;
            turnaround_time[min_index] = time - arrival_time[min_index];
            waiting_time[min_index] = turnaround_time[min_index] - burst_time[min_index];
        }
    }

    // Display Results
    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", id[i], arrival_time[i], burst_time[i], priority[i],
waiting_time[i], turnaround_time[i]);
        total_wt += waiting_time[i];
        total_tat += turnaround_time[i];
    }

    printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
```

}

Output:

```
┌──(student㉿kali)-[~]
└─$ vi prinon.c

┌──(student㉿kali)-[~]
└─$ vi pripre.c

┌──(student㉿kali)-[~]
└─$ gcc pripre.c -o pripre

┌──(student㉿kali)-[~]
└─$ ./pripre
Enter total number of processes: 4
Enter Arrival Time, Burst Time, and Priority for P1: 0 6 3
Enter Arrival Time, Burst Time, and Priority for P2: 0 2 2
Enter Arrival Time, Burst Time, and Priority for P3: 0 14 1
Enter Arrival Time, Burst Time, and Priority for P4: 0 6 4

Process Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
P1      0               6               3               16              22
P2      0               2               2               14              16
P3      0               14              1               0               14
P4      0               6               4               22              28

Average Waiting Time: 13.00
Average Turnaround Time: 20.00

┌──(student㉿kali)-[~]
```

Sample Output:



Result:

Thus,the priority scheduling has been successfully executed.

Ex. No.: 6d)
Date:24.3.25

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:

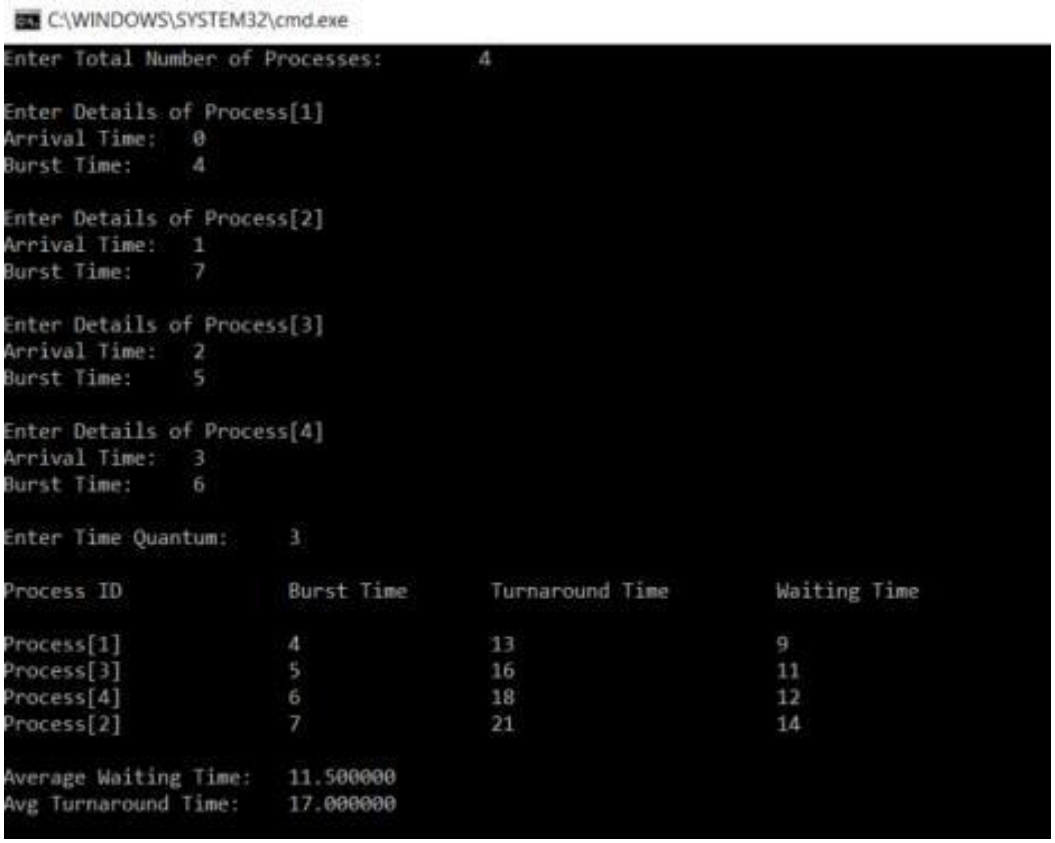
```
#include <stdio.h>


int main() {

   int n, quantum;

   printf("Enter total number of processes: ");

   scanf("%d", &n);
```

```
    int id[n], arrival_time[n], burst_time[n], remaining_time[n], waiting_time[n],
turnaround_time[n];


    for (int i = 0; i < n; i++) {
        id[i] = i + 1;
        printf("Enter Arrival Time and Burst Time for Process[%d]: ", id[i]);
        scanf("%d %d", &arrival_time[i], &burst_time[i]);
        remaining_time[i] = burst_time[i]; // Copy burst time to remaining time
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &quantum);

    int completed = 0, time = 0;
    while (completed < n) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                done = 0;
                if (remaining_time[i] > quantum) {
                    time += quantum;
                    remaining_time[i] -= quantum;
```

```
            } else {
                time += remaining_time[i];
                turnaround_time[i] = time - arrival_time[i];
                waiting_time[i] = turnaround_time[i] - burst_time[i];
                remaining_time[i] = 0;
                completed++;
            }
        }
    }
    if (done) break;
}

// Display Results
printf("\nProcess\tBurst Time\tTurnaround Time\tWaiting Time\n");
float total_wt = 0, total_tat = 0;
for (int i = 0; i < n; i++) {
    printf("P%d\t%d\t\t%d\t\t%d\n", id[i], burst_time[i], turnaround_time[i], waiting_time[i]);
    total_wt += waiting_time[i];
    total_tat += turnaround_time[i];
}

printf("\nAverage Waiting Time: %.6f", total_wt / n);
printf("\nAverage Turnaround Time: %.6f\n", total_tat / n);
```

```
    return 0;
}
```

Sample output:



Result:

Thus the round robin has been successfully executed.