Ex. No.: 6a) **231901039**

Date: 25/2/2025

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>
#include <string.h>
typedef struct {
    char name[10];
    int burst_time;
    int waiting_time;
    int turnaround_time;
} Process;

int main() {
    Process p[10];
    int n;
    float total_wt = 0, total_tat = 0;

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

    for (int i = 0; i < n; i++) {
        printf("\nEnter process %d name: ", i + 1);
        scanf("%s", p[i].name);
        printf("Enter burst time: ");
        scanf("%d", &p[i].burst_time);
    }
    // Calculate waiting time and turnaround time
    p[0].waiting_time = 0;
    p[0].turnaround_time = p[0].burst_time;
    for (int i = 1; i < n; i++) {
        p[i].waiting_time = p[i - 1].waiting_time + p[i -
1].burst_time;
        p[i].turnaround_time = p[i].waiting_time +
p[i].burst_time;
    }
    // Calculate total waiting and turnaround times
    for (int i = 0; i < n; i++) {
        total_wt += p[i].waiting_time;
        total_tat += p[i].turnaround_time;
    }
    // Display results
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%s\t%d\t\t%d\t\t%d\n", p[i].name,
p[i].burst_time, p[i].waiting_time, p[i].turnaround_time);
    }
    printf("\nTotal Waiting Time: %.2f", total_wt);
    printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat /
n);
    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
```

Result:

First-come First- serve (FCFS) scheduling technique has been implemented

Ex. No.: 6b) **231901054**

Date: 01/3/2025

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>
#include <string.h>

typedef struct {
    char name[10];
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
} Process;

void sort_by_burst_time(Process p[], int n) {
    Process temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (p[i].burst_time > p[j].burst_time) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}
```

```
int main() {
    Process p[10];
    int n;
    float avg_wt = 0, avg_tat = 0;

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

    for (int i = 0; i < n; i++) {
        printf("\nProcess %d name: ", i + 1);
        scanf("%s", p[i].name);

        printf("Arrival time: ");
        scanf("%d", &p[i].arrival_time);

        printf("Burst time: ");
        scanf("%d", &p[i].burst_time);

        p[i].waiting_time = 0;
        p[i].turnaround_time = 0;
    }

    // Sort processes by burst time
    sort_by_burst_time(p, n);

    // Calculate waiting time and turnaround time
    p[0].waiting_time = 0;
    p[0].turnaround_time = p[0].burst_time;

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

    // Calculate averages
    for (int i = 0; i < n; i++) {
        avg_wt += p[i].waiting_time;
        avg_tat += p[i].turnaround_time;
    }

    // Display output
    printf("\nProcess\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("%s\t%d\t%d\t%d\t%d\n", p[i].name,
p[i].arrival_time, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
    }

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

Shortest Job First (SJF) scheduling technique has been implemented.

Ex. No.: 6c) **231901054**
Date: 21/3/2025

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

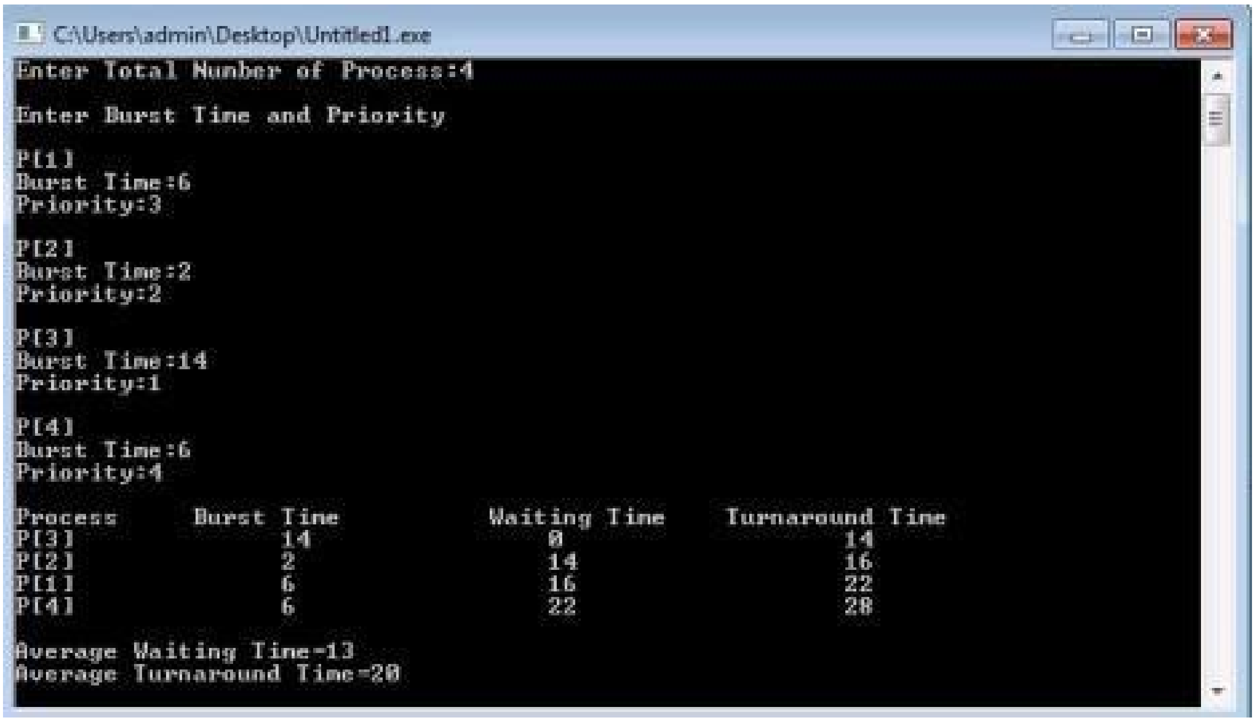
Program Code:

```
#include <stdio.h>
#include <string.h>
#define MAX 100

typedef struct {
    char name[10];
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
} Process;

void sort(Process p[], int n) {
    Process temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (p[i].priority > p[j].priority) {
                // Swap processes based on priority
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}
```

```
int main() {
    Process p[MAX];
    int n;
    float avg_waiting_time = 0, avg_turnaround_time = 0;

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

    for (int i = 0; i < n; i++) {
        printf("\nProcess %d name: ", i + 1);
        scanf("%s", p[i].name);

        printf("Burst time: ");
        scanf("%d", &p[i].burst_time);

        printf("Priority: ");
        scanf("%d", &p[i].priority);

        p[i].waiting_time = 0;
        p[i].turnaround_time = 0;
    }

    // Sort processes based on priority
    sort(p, n);

    // Calculate waiting time and turnaround time
    for (int i = 0; i < n; i++) {
        if (i == 0) {
            p[i].waiting_time = 0;
        } else {
            p[i].waiting_time = p[i-1].waiting_time +
p[i-1].burst_time;
        }
        p[i].turnaround_time = p[i].waiting_time +
p[i].burst_time;

        avg_waiting_time += p[i].waiting_time;
        avg_turnaround_time += p[i].turnaround_time;
    }

    printf("\nProcess\tBT\tPriority\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("%s\t%d\t%d\t\t%d\t%d\n", p[i].name,
p[i].burst_time, p[i].priority, p[i].waiting_time,
p[i].turnaround_time);
```

```
    }

    printf("\nAverage Waiting Time = %.2f", avg_waiting_time /
n);
    printf("\nAverage Turnaround Time = %.2f\n",
avg_turnaround_time / n);

    return 0;
}
```

Sample Output:



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[3]              14                 0                 14
P[2]              2                  14                16
P[1]              6                  16                22
P[4]              6                  22                28

Average Waiting Time=13
Average Turnaround Time=20
```

Result:

The priority scheduling technique has been implemented.

Ex. No.: 6d) **231901054**
Date: 26/3/2025

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>
#include <string.h>
#define MAX 100

typedef struct {
    char name[10];
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
} Process;

int main() {
    Process p[MAX];
    int n, time_quantum, total_time = 0, completed = 0;
    float avg_waiting_time = 0, avg_turnaround_time = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the time quantum: ");
    scanf("%d", &time_quantum);

    for (int i = 0; i < n; i++) {
        printf("\nProcess %d name: ", i + 1);
        scanf("%s", p[i].name);

        printf("Arrival time: ");
        scanf("%d", &p[i].arrival_time);

        printf("Burst time: ");
        scanf("%d", &p[i].burst_time);

        p[i].remaining_time = p[i].burst_time;
        p[i].waiting_time = 0;
        p[i].turnaround_time = 0;
    }

    int time = 0;
    while (completed != n) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_time > 0 && p[i].arrival_time
<= time) {
                done = 0;

                if (p[i].remaining_time > time_quantum) {
                    time += time_quantum;
                    p[i].remaining_time -= time_quantum;
                } else {
                    time += p[i].remaining_time;
                    p[i].waiting_time = time -
p[i].arrival_time - p[i].burst_time;
                    p[i].turnaround_time = time -
p[i].arrival_time;
                    p[i].remaining_time = 0;
                    completed++;
                }
            }
        }
        if (done) {
            time++; //If no process is ready,move forward in
time
        }
    }

    printf("\nProcess\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("%s\t%d\t%d\t%d\t%d\n", p[i].name,
p[i].arrival_time, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
        avg_waiting_time += p[i].waiting_time;
        avg_turnaround_time += p[i].turnaround_time;
    }

    printf("\nAverage Waiting Time = %.2f", avg_waiting_time /
n);
    printf("\nAverage Turnaround Time = %.2f\n",
avg_turnaround_time / n);

    return 0;
}
```

Sample Output:

```
Enter the number of processes: 5
Enter the time quantum: 2

Process 1 name: 1
Arrival time: 0
Burst time: 3

Process 2 name: 2
Arrival time: 3
Burst time: 3

Process 3 name: 3
Arrival time: 5
Burst time: 2

Process 4 name: 4
Arrival time: 6
Burst time: 4

Process 5 name: 5
Arrival time: 8
Burst time: 5
```

```
Process   AT   BT   WT   TAT
1         0    3    0    3
2         3    3    6    9
3         5    2    0    2
4         6    4    4    8
5         8    5    4    9

Average Waiting Time = 2.80
Average Turnaround Time = 6.20
```

Result:

The Round Robin (RR) scheduling technique has been implemented.