Ex. No.: 6b)

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-PREMPTION

```
#include <stdio.h>

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

void sortProcesses(struct Process p[], int n) { struct 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; } } } }

void calculateNonPreemptiveSJF(struct Process p[], int n) { int total_wt = 0, total_tat = 0;

// Sort processes based on burst time for non-preemptive SJF
sortProcesses(p, n);

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

```
}

for (int i = 0; i < n; i++) {
    p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
    total_wt += p[i].waiting_time;
    total_tat += p[i].turnaround_time;
}

// Display results
printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%s\t\t%d\t\t%d\t\t%d\t\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", (float)total_wt / n);
printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);


}

int main() { int n;

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

struct Process p[n];

// Input process details
for (int i = 0; i < n; i++) {
    printf("\nEnter process name (e.g., P1, P2, etc.): ");
    scanf("%s", p[i].name);
    printf("Enter arrival time for %s: ", p[i].name);
    scanf("%d", &p[i].arrival_time);
    printf("Enter burst time for %s: ", p[i].name);
    scanf("%d", &p[i].burst_time);
}

// Calculate and display Non-preemptive SJF results
calculateNonPreemptiveSJF(p, n);
```

```
return 0;

}
```

OUTPUT:

Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P4	3	3	0	3
P1	0	6	3	9
P3	2	7	9	16
P2	1	8	16	24

Average Waiting Time: 7.00

Average Turnaround Time: 13.00

PREMPTION

```
#include <stdio.h>

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

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

void calculatePreemptiveSJF(struct Process p[], int n) { int total_wt = 0, total_tat = 0; int completed = 0, current_time = 0;
```

```
// Sort processes based on arrival time first
sortProcesses(p, n);

// Initialize remaining time for each process
for (int i = 0; i < n; i++) {
    p[i].remaining_time = p[i].burst_time;
}

// Run the preemptive SJF (SRTF)
while (completed < n) {
    int shortest = -1;
    int min_remaining_time = 100000;

    // Find the process with the shortest remaining time that has arrived
    for (int i = 0; i < n; i++) {
        if (p[i].arrival_time <= current_time && p[i].remaining_time > 0 &&
p[i].remaining_time < min_remaining_time) {
            min_remaining_time = p[i].remaining_time;
            shortest = i;
        }
    }

    if (shortest != -1) {
        // Process the shortest job
        p[shortest].remaining_time--;
        current_time++;

        // If the process has completed
        if (p[shortest].remaining_time == 0) {
            p[shortest].turnaround_time = current_time - p[shortest].arrival_time;
            p[shortest].waiting_time = p[shortest].turnaround_time - p[shortest].burst_time;
            total_wt += p[shortest].waiting_time;
            total_tat += p[shortest].turnaround_time;
            completed++;
        }
    } else {
        current_time++; // No process can be executed, so move time forward
    }
}
```

```
// Display results
printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%s\t\t%d\t\t%d\t\t%d\t\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", (float)total_wt / n);
printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);


}

int main() { int n;

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

struct Process p[n];

// Input process details
for (int i = 0; i < n; i++) {
    printf("\nEnter process name (e.g., P1, P2, etc.): ");
    scanf("%s", p[i].name);
    printf("Enter arrival time for %s: ", p[i].name);
    scanf("%d", &p[i].arrival_time);
    printf("Enter burst time for %s: ", p[i].name);
    scanf("%d", &p[i].burst_time);
}

// Calculate and display Preemptive SJF (SRTF) results
calculatePreemptiveSJF(p, n);

return 0;


}
```

OUTPUT:

```
Process    Arrival Time    Burst Time    Waiting Time    Turnaround Time
```

P4	3	3	0	3
P1	0	6	3	9
P3	2	7	9	16
P2	1	8	16	24

Average Waiting Time: 7.00

Average Turnaround Time: 13.00

RESULT:

Program to implement the Shortest Job First (SJF) scheduling technique is successful.