

Experiment-7:Construct a C program to implement a non-preemptive SJF algorithm.

Aim:

The aim of this program is to implement the Non-Preemptive Shortest Job First (SJF) Scheduling Algorithm in C. In the non-preemptive SJF algorithm, the process with the shortest burst time is selected for execution first, and once a process starts executing, it runs to completion without being interrupted.

Procedure:

1. Input:
 - Number of processes.
 - Burst time for each process.
2. Sorting:
 - Sort processes in ascending order of burst time. In case two processes have the same burst time, they are processed based on their arrival order.
3. Execution:
 - Select the process with the shortest burst time from the ready queue and execute it.
4. Waiting Time Calculation:
 - Calculate the waiting time for each process. Waiting time is the total time a process spends waiting in the ready queue before it gets executed.
5. Turnaround Time Calculation:
 - Calculate the turnaround time for each process. Turnaround time is the total time taken from the arrival of the process to its completion.
6. Output:
 - Output the process ID, burst time, waiting time, and turnaround time for each process, as well as the average waiting time and average turnaround time.

Non-Preemptive Shortest Job First (SJF) Scheduling Algorithm:

- Non-preemptive SJF means that once a process starts executing, it runs to completion.
- Shortest Job First selects the process with the shortest burst time to execute next.

C Program Implementation:

```
#include <stdio.h>

struct Process {

    int id;

    int burst_time;

    int waiting_time;

    int turnaround_time;

};

void sortByBurstTime(struct Process processes[], int n) {

    struct Process temp;

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (processes[i].burst_time > processes[j].burst_time) {

                // Swap processes[i] and processes[j]

                temp = processes[i];

                processes[i] = processes[j];

                processes[j] = temp;

            }

        }

    }

}

int main() {

    int n;

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

    scanf("%d", &n);

    struct Process processes[n];

    int total_waiting_time = 0, total_turnaround_time = 0;
```

```

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

    processes[i].id = i + 1; // Assign process ID

    printf("Enter burst time for process %d: ", i + 1);

    scanf("%d", &processes[i].burst_time);

}

sortByBurstTime(processes,

processes[0].waiting_time = 0; // The first process has no waiting time

for (int i = 1; i < n; i++) {

    processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;

}

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

    processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;

    total_waiting_time += processes[i].waiting_time;

    total_turnaround_time += processes[i].turnaround_time;

}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

    printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time,
processes[i].waiting_time,

        processes[i].turnaround_time);

}

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

printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);

return 0;

}

```

Output:

Output

```
Enter the number of processes: 2
Enter burst time for process 1: 4
Enter burst time for process 2: 6
```

Process	Burst Time	Waiting Time	Turnaround Time
1	4	0	4
2	6	4	10

```
Average Waiting Time: 2.00
```

```
Average Turnaround Time: 7.00
```