

EXPERIMENT-7

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

AIM:-

To design a C program to implement the **Non-Preemptive Shortest Job First (SJF) Scheduling Algorithm**, which selects the process with the shortest burst time from the set of available processes and executes it until completion.

ALGORITHM:-

Input Process Details:

Read the number of processes.

Input the arrival time and burst time for each process.

Sort Processes Based on Arrival Time and Burst Time:

Sort the processes such that the shortest burst time process among those that have arrived is selected first.

Execute the Processes:

For each selected process, update the completion time, turnaround time, and waiting time.

Calculate Turnaround Time (TAT):

$TAT = \text{Completion Time} - \text{Arrival Time}$.

Calculate Waiting Time (WT):

$WT = TAT - \text{Burst Time}$.

Calculate Averages:

Compute the average turnaround time and waiting time.

Display Results:

Show process details, including burst time, arrival time, completion time, turnaround time, and waiting time.

CODE:-

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

```
typedef struct {
```

```
    int process_id;
```

```
    int arrival_time;
```

```
    int burst_time;
```

```
    int completion_time;
```

```
    int turnaround_time;
```

```
    int waiting_time;
```

```
    int visited; // Flag to check if the process is executed
```

```
} Process;
```

```
int main() {
```

```
    int n, current_time = 0, completed = 0;
```

```
    float avg_turnaround_time = 0, avg_waiting_time = 0;
```

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

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

```
    Process processes[n];
```

```
// Input process details
```

```
printf("Enter the arrival time and burst time for each process:\n");
```

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

```
    processes[i].process_id = i + 1;
```

```
    printf("Process %d - Arrival Time: ", i + 1);
```

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

```
    printf("Process %d - Burst Time: ", i + 1);
```

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

```
    processes[i].visited = 0; // Initially, no process is executed
```

```
}
```

```
// Non-Preemptive SJF Scheduling
```

```
while (completed != n) {
```

```
    int shortest_job_index = -1;
```

```
    int shortest_burst_time = 1e9; // Large value for comparison
```

```
// Find the process with the shortest burst time among the arrived processes
```

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

```
    if (processes[i].arrival_time <= current_time && !processes[i].visited) {
```

```
        if (processes[i].burst_time < shortest_burst_time ||
```

```
            (processes[i].burst_time == shortest_burst_time && processes[i].arrival_time <
processes[shortest_job_index].arrival_time)) {
```

```
                shortest_burst_time = processes[i].burst_time;
```

```
                shortest_job_index = i;
```

```
        }
```

```

    }

}

if (shortest_job_index != -1) {

    // Execute the selected process

    current_time += processes[shortest_job_index].burst_time;

    processes[shortest_job_index].completion_time = current_time;

    processes[shortest_job_index].turnaround_time =
processes[shortest_job_index].completion_time - processes[shortest_job_index].arrival_time;

    processes[shortest_job_index].waiting_time =
processes[shortest_job_index].turnaround_time - processes[shortest_job_index].burst_time;


    // Update averages

    avg_turnaround_time += processes[shortest_job_index].turnaround_time;

    avg_waiting_time += processes[shortest_job_index].waiting_time;


    // Mark process as completed

    processes[shortest_job_index].visited = 1;

    completed++;

} else {

    // If no process has arrived, increment the current time

    current_time++;

}

}

```

```

// Calculate averages

avg_turnaround_time /= n;

avg_waiting_time /= n;


// Display results

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

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

    printf("%d\t%d\t%d\t%d\t%d\t%d\n",                processes[i].process_id,
processes[i].arrival_time,

    processes[i].burst_time, processes[i].completion_time,

    processes[i].turnaround_time, processes[i].waiting_time);

}

printf("\nAverage Turnaround Time: %.2f\n", avg_turnaround_time);

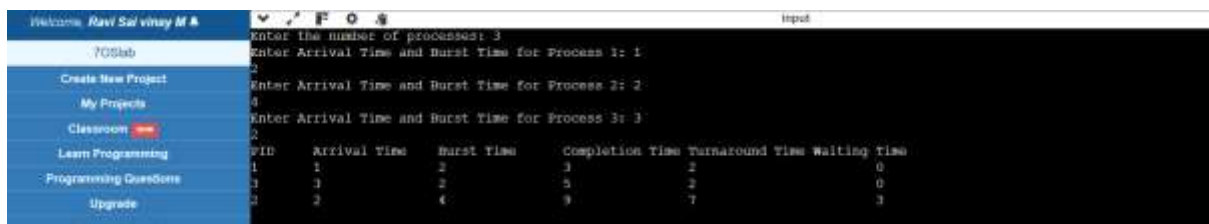
printf("Average Waiting Time: %.2f\n", avg_waiting_time);

return 0;

}

```

OUTPUT:-



```

Welcome, Ravi Sai vinay M #
?CSLab
Create New Project
My Projects
Classroom
Learn Programming
Programming Questions
Upgrade

Enter the number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 1
3
Enter Arrival Time and Burst Time for Process 2: 2
3
Enter Arrival Time and Burst Time for Process 3: 3
3
PID    Arrival Time    Burst Time    Completion Time    Turnaround Time    waiting Time
1      1                2              3                  2                  0
2      3                2              5                  2                  0
3      2                4              6                  4                  2

```

RESULT:-

The Non-Preemptive Shortest Job First (SJF) scheduling algorithm was successfully implemented in C. The program accurately calculates the completion, turnaround, and waiting times for all processes and displays the results along with the average turnaround and waiting times.