B. Siva Shirish -192324016

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

Aim:

To design a C program that implements a non-preemptive Shortest Job First (SJF) scheduling algorithm, where the process with the shortest burst time is selected next for execution.

Algorithm:

1. Start the program.
2. Input the number of processes, their burst times, and arrival times.
3. Initialize variables for tracking time and process completion.
4. At each time unit:
    - Check all arrived and non-completed processes.
    - Select the process with the shortest burst time.
    - Execute the selected process to completion.
5. Record the completion time, waiting time, and turnaround time for each process.
6. Repeat until all processes are completed.
7. Calculate average waiting time and turnaround time.
8. Display the results.
9. End the program.

Procedure:

1. Include necessary headers: <stdio.h> and <limits.h> (for constants like INT_MAX).
2. Use arrays to store process attributes such as burst times, arrival times, waiting times, and turnaround times.
3. Implement a loop to simulate the scheduling and dynamically select the shortest job.
4. Track completion and compute metrics.

CODE:

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

int main() {
   int n, i, completed = 0, time = 0, min_burst, current_process = -1;
   float avg_wait = 0, avg_turnaround = 0;

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

   int burst_time[n], arrival_time[n], waiting_time[n], turnaround_time[n],
completion_time[n];
   int is_completed[n];

   printf("Enter arrival times and burst times for each process:\n");
   for (i = 0; i < n; i++) {
      printf("Process %d:\n", i + 1);
      printf("Arrival Time: ");
      scanf("%d", &arrival_time[i]);
      printf("Burst Time: ");
      scanf("%d", &burst_time[i]);
      is_completed[i] = 0;
   }

   while (completed < n) {
      min_burst = INT_MAX;
      current_process = -1;
```

```
    for (i = 0; i < n; i++) {
      if (arrival_time[i] <= time && !is_completed[i] && burst_time[i] < min_burst) {
        min_burst = burst_time[i];
        current_process = i;
      }
    }

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

    time += burst_time[current_process];
    completion_time[current_process] = time;
    turnaround_time[current_process] = completion_time[current_process] -
arrival_time[current_process];
    waiting_time[current_process] = turnaround_time[current_process] -
burst_time[current_process];

    avg_wait += waiting_time[current_process];
    avg_turnaround += turnaround_time[current_process];
    is_completed[current_process] = 1;
    completed++;
  }

  avg_wait /= n;
  avg_turnaround /= n;
```

```
    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, arrival_time[i], burst_time[i],
waiting_time[i], turnaround_time[i]);
    }

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

    return 0;
}
```

OUTPUT: