LAB Assignment 5
Submitted by – **Isha Gupta**
Roll no: **102303007**
Subgroup 2C11

Q1. Write a program using C/C++/Java to simulate the FCFS, SJF (pre-emptive as well as non preemptive approach). The scenario is: user may input n processes with respective CPU burst time and arrival time. System will ask the user to select the type of algorithm from the list mentioned above. System should display the waiting time for each process, average waiting time for the whole system, and final execution sequence.

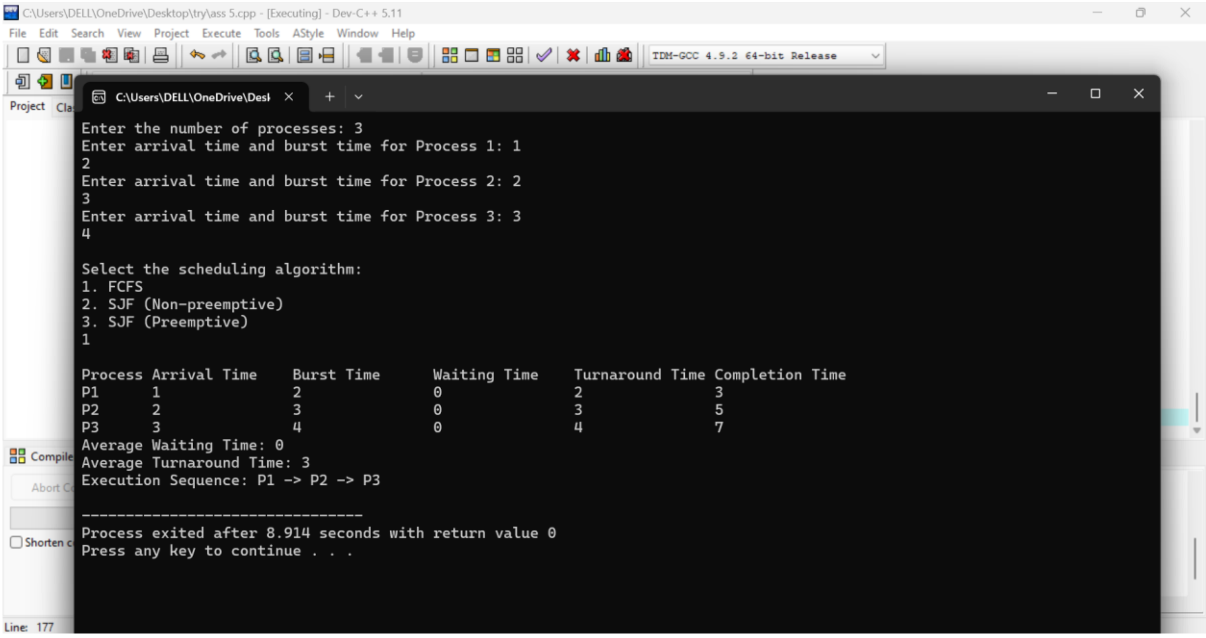
```
#include <iostream>
#include <iomanip>
#include <limits>

using namespace std;

struct Process {
    int id; // Process ID
    int arrival_time; // Arrival time of the process
    int burst_time; // Burst time of the process
    int waiting_time; // Waiting time of the process
    int turnaround_time; // Turnaround time of the process
    int completion_time; // Completion time of the process
};

// Function to calculate waiting times and turnaround times for FCFS
void calculateFCFS(Process processes[], int n) {
    processes[0].waiting_time = 0; // First process has no waiting time

    for (int i = 1; i < n; i++) {
        processes[i].waiting_time = processes[i - 1].completion_time - processes[i].arrival_time;
        if (processes[i].waiting_time < 0) {
            processes[i].waiting_time = 0; // If it arrives after the previous process completes
        }
    }

    // Calculate turnaround time and completion time
    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        processes[i].completion_time = processes[i].arrival_time + processes[i].waiting_time +
processes[i].burst_time;
    }
}

// Function to calculate waiting times and turnaround times for SJF (Non-preemptive)
```

```
void calculateSJFNonPreemptive(Process processes[], int n) {
    bool completed[100] = {false}; // Track completed processes
    int current_time = 0;
    int completed_processes = 0;

    while (completed_processes < n) {
        int idx = -1;
        int min_burst_time = numeric_limits<int>::max();

        // Find the process with the shortest burst time that has arrived
        for (int i = 0; i < n; i++) {
            if (!completed[i] && processes[i].arrival_time <= current_time) {
                if (processes[i].burst_time < min_burst_time) {
                    min_burst_time = processes[i].burst_time;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            processes[idx].waiting_time = current_time - processes[idx].arrival_time;
            if (processes[idx].waiting_time < 0) {
                processes[idx].waiting_time = 0; // If it arrives after the current time
            }
            current_time += processes[idx].burst_time;
            processes[idx].completion_time = current_time;
            processes[idx].turnaround_time = processes[idx].waiting_time +
processes[idx].burst_time;
            completed[idx] = true;
            completed_processes++;
        } else {
            current_time++; // No process is ready, increment current time
        }
    }
}

// Function to calculate waiting times and turnaround times for SJF (Preemptive)
void calculateSJFPreemptive(Process processes[], int n) {
    int remaining_time[100]; // Store remaining time for each process
    for (int i = 0; i < n; i++) {
        remaining_time[i] = processes[i].burst_time;
    }

    int current_time = 0;
    int completed_processes = 0;

    while (completed_processes < n) {
        int idx = -1;
```

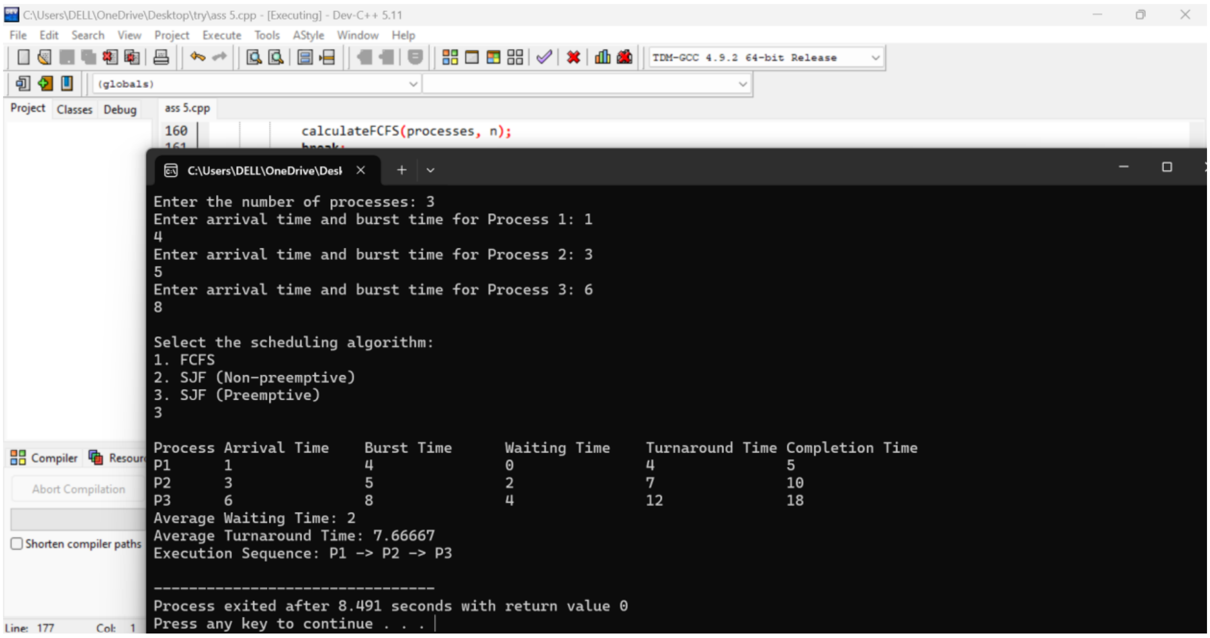
```
        int min_burst_time = numeric_limits<int>::max();

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

        if (idx != -1) {
            remaining_time[idx]--;
            if (remaining_time[idx] == 0) {
                processes[idx].completion_time = current_time + 1;
                processes[idx].turnaround_time = processes[idx].completion_time -
processes[idx].arrival_time;
                processes[idx].waiting_time = processes[idx].turnaround_time -
processes[idx].burst_time;
                completed_processes++;
            }
        }

        current_time++;
    }
}

// Function to display results
void displayResults(Process processes[], int n) {
    double total_waiting_time = 0;
    double total_turnaround_time = 0;

    cout << "\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\tCompletion Time\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << processes[i].id << "\t"
             << processes[i].arrival_time << "\t\t"
             << processes[i].burst_time << "\t\t"
             << processes[i].waiting_time << "\t\t"
             << processes[i].turnaround_time << "\t\t"
             << processes[i].completion_time << "\n";
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Calculate and display average waiting time and average turnaround time
    cout << "Average Waiting Time: " << total_waiting_time / n << "\n";
```

```
    cout << "Average Turnaround Time: " << total_turnaround_time / n << "\n";

    // Display the execution sequence
    cout << "Execution Sequence: ";
    for (int i = 0; i < n; i++) {
        cout << "P" << processes[i].id;
        if (i < n - 1) {
            cout << " -> ";
        }
    }
    cout << "\n";
}

// Main function
int main() {
    int n, choice;
    cout << "Enter the number of processes: ";
    cin >> n;

    Process processes[100]; // Array to hold processes

    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1; // Process ID
        cout << "Enter arrival time and burst time for Process " << (i + 1) << ": ";
        cin >> processes[i].arrival_time >> processes[i].burst_time;
    }

    cout << "\nSelect the scheduling algorithm:\n";
    cout << "1. FCFS\n2. SJF (Non-preemptive)\n3. SJF (Preemptive)\n";
    cin >> choice;

    switch (choice) {
        case 1:
            calculateFCFS(processes, n);
            break;
        case 2:
            calculateSJFNonPreemptive(processes, n);
            break;
        case 3:
            calculateSJFPreemptive(processes, n);
            break;
        default:
            cout << "Invalid choice\n";
            return 1;
    }

    displayResults(processes, n);
```

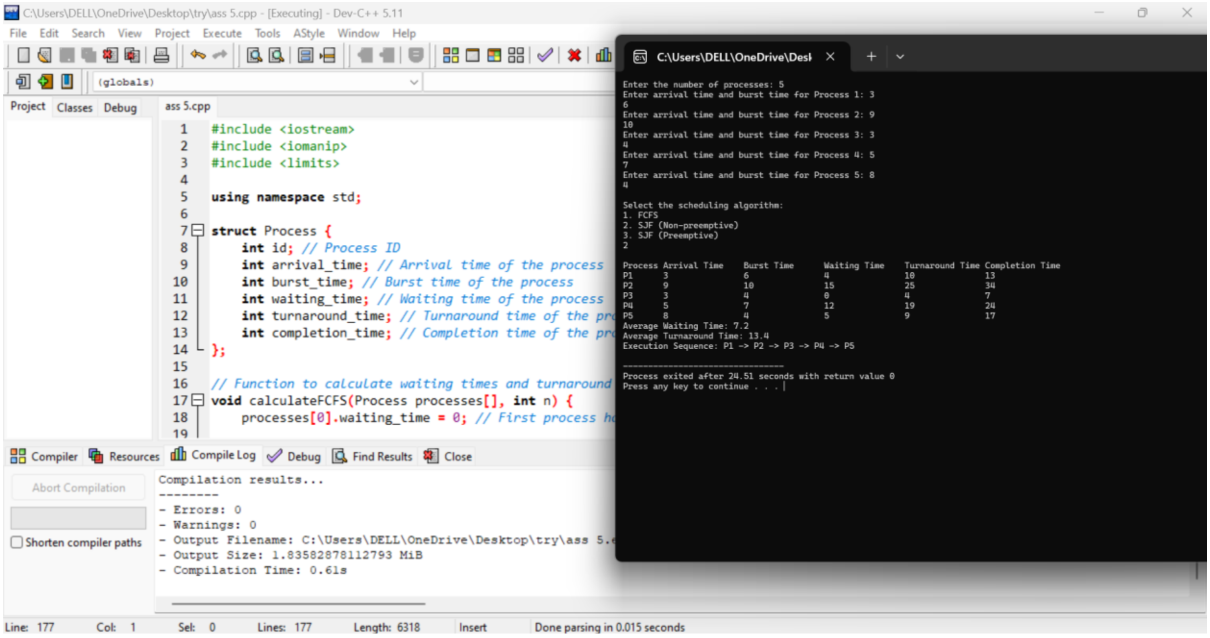
```
    return 0;
}
```


C:\Users\DELL\OneDrive\Desktop\try\ass 5.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
Enter the number of processes: 3
Enter arrival time and burst time for Process 1: 1
2
Enter arrival time and burst time for Process 2: 2
3
Enter arrival time and burst time for Process 3: 3
4
Select the scheduling algorithm:
1. FCFS
2. SJF (Non-preemptive)
3. SJF (Preemptive)
1
Process Arrival Time Burst Time Waiting Time Turnaround Time Completion Time
P1 1 2 0 2 3
P2 2 3 0 3 5
P3 3 4 0 4 7
Average Waiting Time: 0
Average Turnaround Time: 3
Execution Sequence: P1 -> P2 -> P3

Process exited after 8.914 seconds with return value 0
Press any key to continue . . .


C:\Users\DELL\OneDrive\Desktop\try\ass 5.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
160 calculateFCFS(processes, n);
Enter the number of processes: 3
Enter arrival time and burst time for Process 1: 1
4
Enter arrival time and burst time for Process 2: 3
5
Enter arrival time and burst time for Process 3: 6
8
Select the scheduling algorithm:
1. FCFS
2. SJF (Non-preemptive)
3. SJF (Preemptive)
3
Process Arrival Time Burst Time Waiting Time Turnaround Time Completion Time
P1 1 4 0 4 5
P2 3 5 2 7 10
P3 6 8 4 12 18
Average Waiting Time: 2
Average Turnaround Time: 7.66667
Execution Sequence: P1 -> P2 -> P3

Process exited after 8.491 seconds with return value 0
Press any key to continue . . .



C:\Users\DELL\OneDrive\Desktop\try\ass 5.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug
ass 5.cpp
#include <iostream>
#include <iomanip>
#include <limits>
using namespace std;
struct Process {
int id; // Process ID
int arrival_time; // Arrival time of the process
int burst_time; // Burst time of the process
int waiting_time; // Waiting time of the process
int turnaround_time; // Turnaround time of the pr
int completion_time; // Completion time of the pr
};
// Function to calculate waiting times and turnaround
void calculateFCFS(Process processes[], int n) {
processes[0].waiting_time = 0; // First process h
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\DELL\OneDrive\Desktop\try\ass 5.
- Output Size: 1.83582878112793 MiB
- Compilation Time: 0.61s
Line: 177 Col: 1 Sel: 0 Lines: 177 Length: 6318 Insert Done parsing in 0.015 seconds
C:\Users\DELL\OneDrive\Desk
Enter the number of processes: 5
Enter arrival time and burst time for Process 1: 3
6
Enter arrival time and burst time for Process 2: 9
10
Enter arrival time and burst time for Process 3: 3
4
Enter arrival time and burst time for Process 4: 5
7
Enter arrival time and burst time for Process 5: 8
4
Select the scheduling algorithm:
1. FCFS
2. SJF (Non-preemptive)
3. SJF (Preemptive)
2
Process Arrival Time Burst Time Waiting Time Turnaround Time Completion Time
P1 3 6 4 10 13
P2 9 10 15 25 34
P3 3 4 0 4 7
P4 5 7 12 19 24
P5 8 4 5 9 17
Average Waiting Time: 7.2
Average Turnaround Time: 13.4
Execution Sequence: P1 -> P2 -> P3 -> P4 -> P5

Process exited after 24.51 seconds with return value 0
Press any key to continue . . .