University of Asia Pacific

Department of Computer Science and Engineering

CSE 402: Operating System Lab

LAB REPORT

Lab Report: 01
Experiment tittle: CPU Scheduling Algorithms Lab Report

Submitted by:

Name : H. M. Tahsin Sheikh

Student ID : 22201243

Section : E1

Submitted to:

Bidita Sarkar Diba

Lecturer,
Department of Computer Science and Engineering

Date of Submission: 25/02/2026

Experiment 1: First Come First Serve (FCFS) Scheduling Algorithm

1. Title of the Experiment

Implementation of First Come First Serve (FCFS) CPU Scheduling Algorithm

2. Objective

• To understand the working principle of the FCFS scheduling algorithm.
• To implement the FCFS algorithm for process scheduling based on arrival time.
• To calculate completion time, turnaround time, and waiting time.
• To analyze the performance of FCFS scheduling.

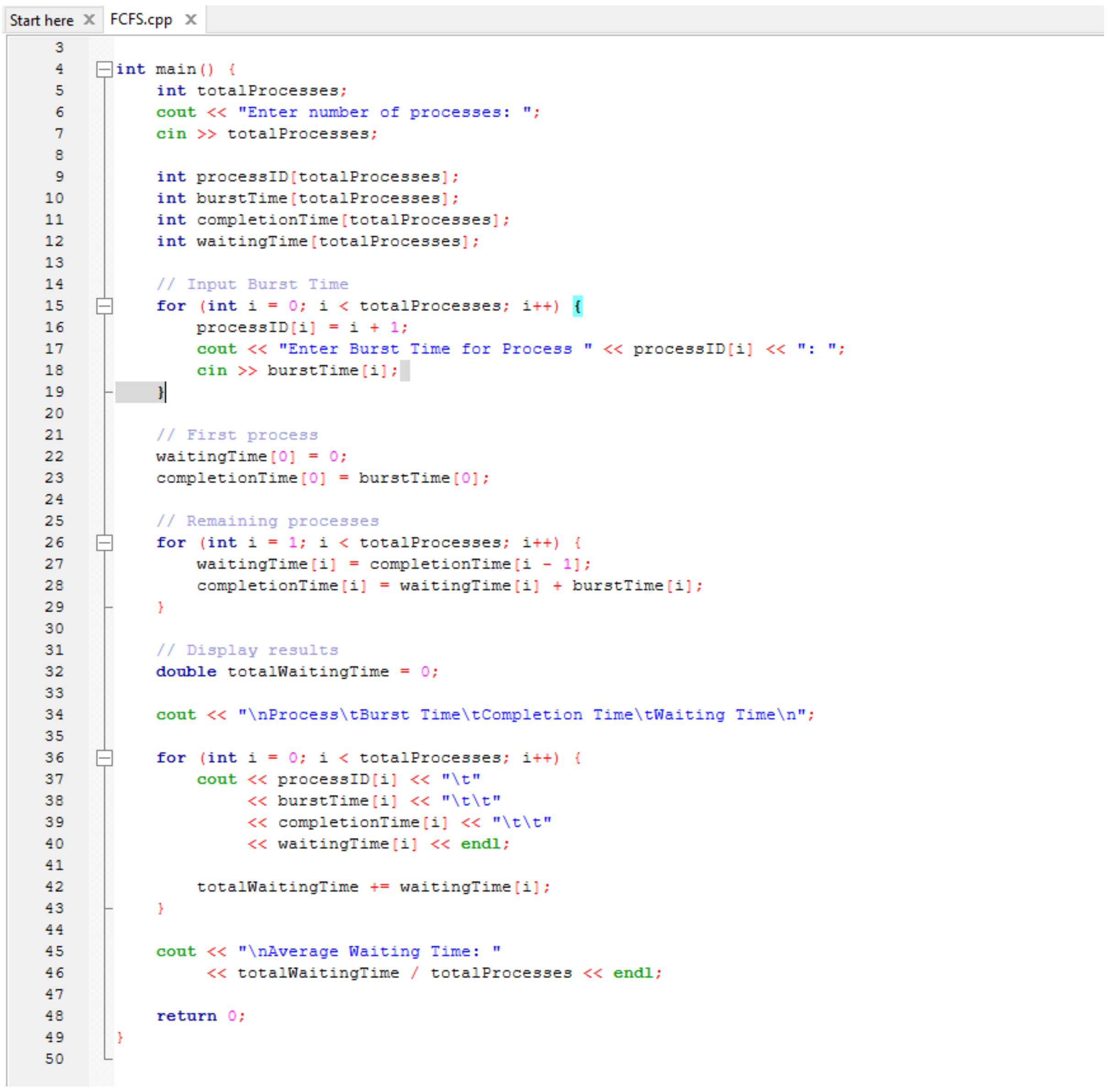
3. Introduction

First Come First Serve (FCFS) is the simplest CPU scheduling algorithm in which processes are executed in the order they arrive in the ready queue. It is a non-preemptive scheduling algorithm, meaning once a process starts execution, it continues until completion. FCFS is easy to implement and ensures fairness, but it may suffer from the convoy effect where short processes wait behind long ones.

4. Algorithm

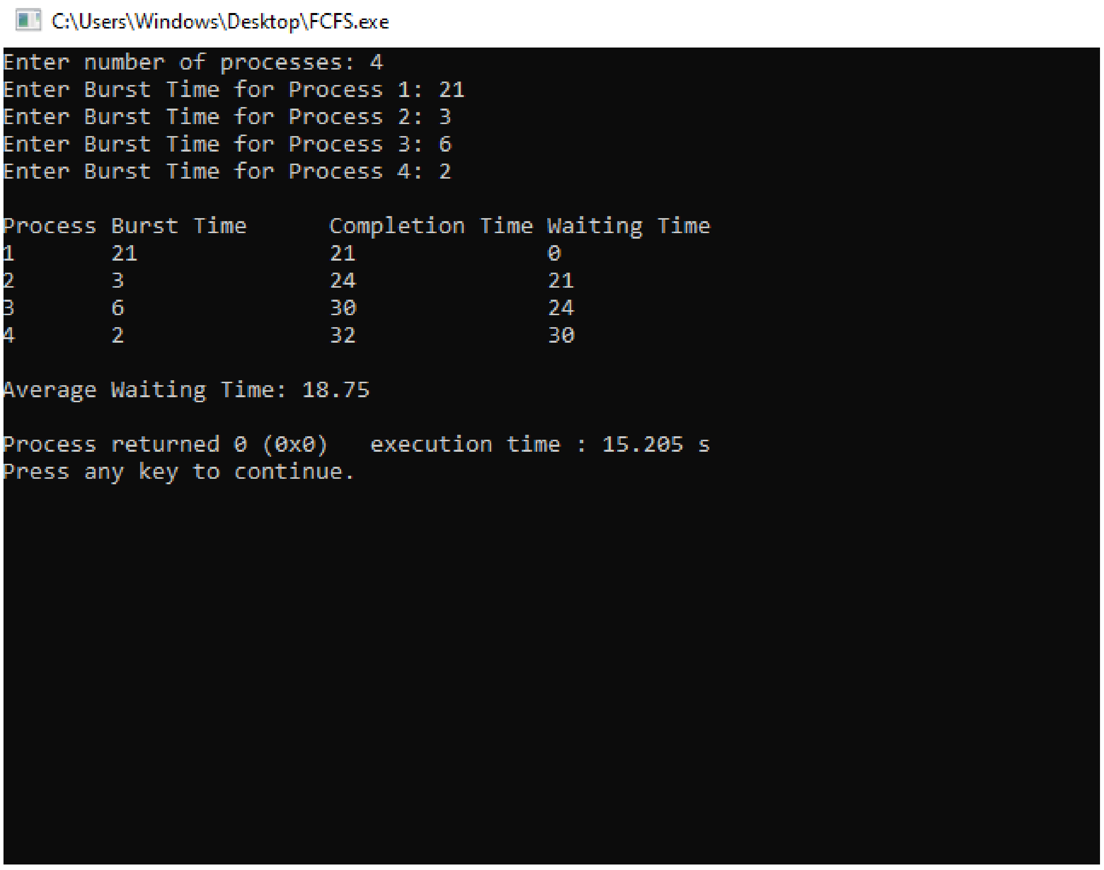
```
Step 1: Input the number of processes.
Step 2: Input arrival time and burst time for each process.
Step 3: Sort processes according to arrival time.
Step 4: Calculate completion time for each process.
Step 5: Calculate turnaround time = completion time – arrival time.
Step 6: Calculate waiting time = turnaround time – burst time.
Step 7: Display results.
```

5. Source Code



```cpp

int main() {
    int totalProcesses;
    cout << "Enter number of processes: ";
    cin >> totalProcesses;

    int processID[totalProcesses];
    int burstTime[totalProcesses];
    int completionTime[totalProcesses];
    int waitingTime[totalProcesses];

    // Input Burst Time
    for (int i = 0; i < totalProcesses; i++) {
        processID[i] = i + 1;
        cout << "Enter Burst Time for Process " << processID[i] << ": ";
        cin >> burstTime[i];
    }

    // First process
    waitingTime[0] = 0;
    completionTime[0] = burstTime[0];

    // Remaining processes
    for (int i = 1; i < totalProcesses; i++) {
        waitingTime[i] = completionTime[i - 1];
        completionTime[i] = waitingTime[i] + burstTime[i];
    }

    // Display results
    double totalWaitingTime = 0;

    cout << "\nProcess\tBurst Time\tCompletion Time\tWaiting Time\n";

    for (int i = 0; i < totalProcesses; i++) {
        cout << processID[i] << "\t"
             << burstTime[i] << "\t\t"
             << completionTime[i] << "\t\t"
             << waitingTime[i] << endl;

        totalWaitingTime += waitingTime[i];
    }

    cout << "\nAverage Waiting Time: "
         << totalWaitingTime / totalProcesses << endl;

    return 0;
}

```

6. Input/Output



C:\Users\Windows\Desktop\FCFS.exe

```
Enter number of processes: 4
Enter Burst Time for Process 1: 21
Enter Burst Time for Process 2: 3
Enter Burst Time for Process 3: 6
Enter Burst Time for Process 4: 2

Process Burst Time      Completion Time Waiting Time
1       21              21              0
2       3               24              21
3       6               30              24
4       2               32              30

Average Waiting Time: 18.75

Process returned 0 (0x0)   execution time : 15.205 s
Press any key to continue.
```

7. Conclusion

The FCFS scheduling algorithm is simple and easy to implement. It ensures that processes are executed in the order of arrival, providing fairness. However, it may lead to higher waiting times and inefficient CPU utilization when long processes arrive before shorter ones.

Experiment 2: Shortest Job First (SJF)Scheduling Algorithm

1. Title of the Experiment

Implementation of Shortest Job First (SJF) CPU Scheduling Algorithm

2. Objective

• To understand the working principle of the SJF scheduling algorithm.
• To implement SJF scheduling based on burst time.
• To calculate completion time, turnaround time, and waiting time.
• To analyze performance compared to FCFS.

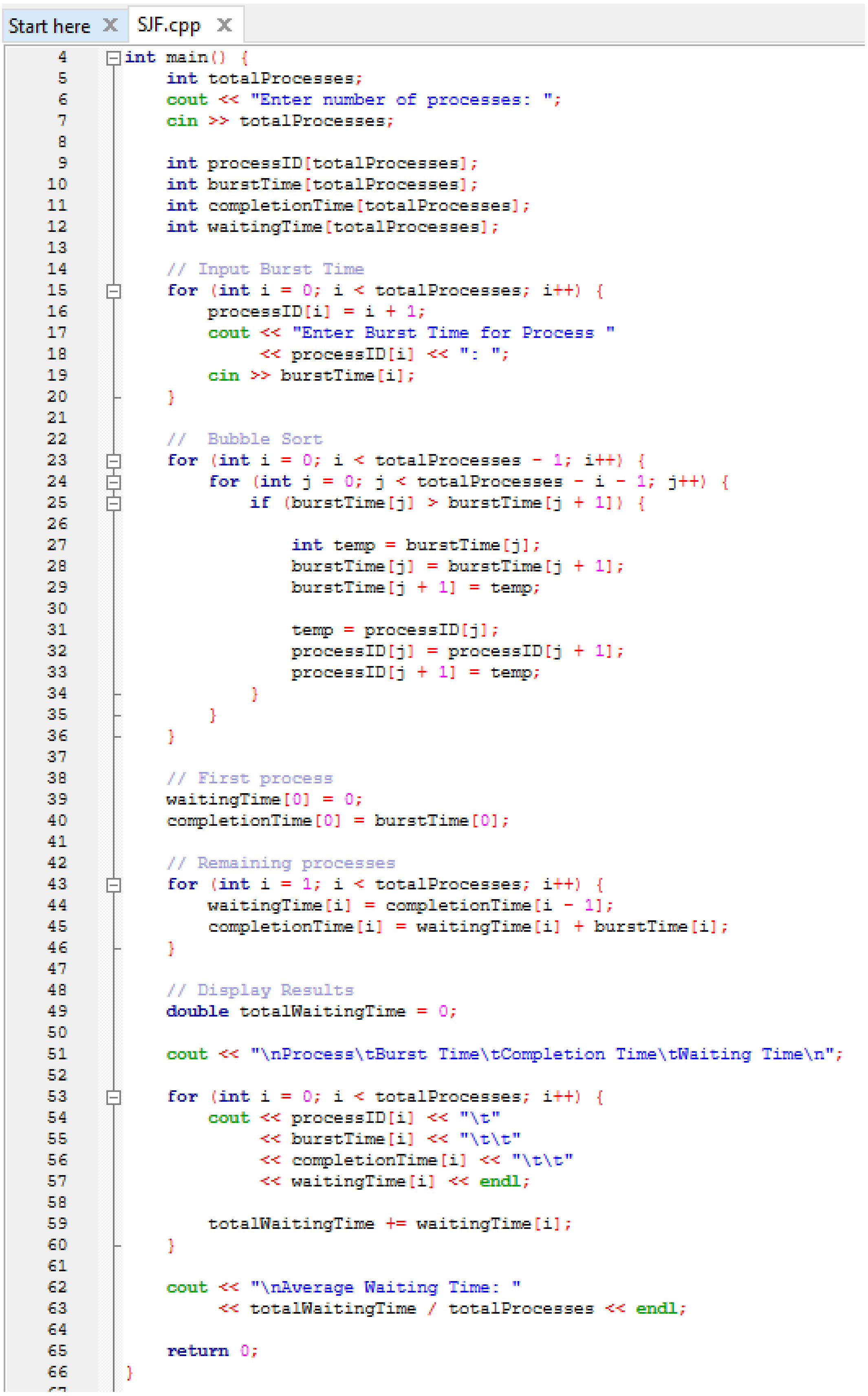
3. Introduction

Shortest Job First (SJF) is a CPU scheduling algorithm in which the process with the smallest burst time is executed first. It can be either preemptive or non-preemptive. SJF provides minimum average waiting time among scheduling algorithms but requires prior knowledge of burst time, which is not always practical.

4. Algorithm

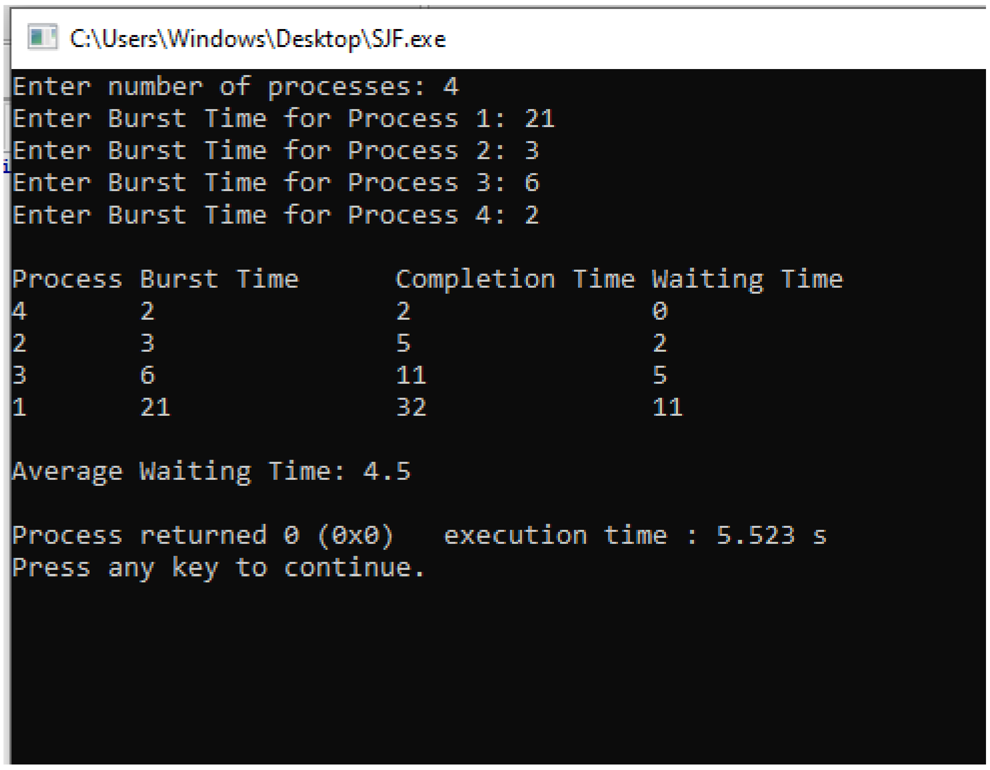
Step 1: Input the number of processes.
Step 2: Input arrival time and burst time for each process.
Step 3: Select the process with the smallest burst time among available processes.
Step 4: Execute the selected process completely (non-preemptive).
Step 5: Update current time and repeat selection for remaining processes.
Step 6: Calculate completion time, turnaround time, and waiting time.
Step 7: Display results.

5. Source Code



```
int main() {
    int totalProcesses;
    cout << "Enter number of processes: ";
    cin >> totalProcesses;

    int processID[totalProcesses];
    int burstTime[totalProcesses];
    int completionTime[totalProcesses];
    int waitingTime[totalProcesses];

    // Input Burst Time
    for (int i = 0; i < totalProcesses; i++) {
        processID[i] = i + 1;
        cout << "Enter Burst Time for Process "
             << processID[i] << ": ";
        cin >> burstTime[i];
    }

    //  Bubble Sort
    for (int i = 0; i < totalProcesses - 1; i++) {
        for (int j = 0; j < totalProcesses - i - 1; j++) {
            if (burstTime[j] > burstTime[j + 1]) {

                int temp = burstTime[j];
                burstTime[j] = burstTime[j + 1];
                burstTime[j + 1] = temp;

                temp = processID[j];
                processID[j] = processID[j + 1];
                processID[j + 1] = temp;
            }
        }
    }

    // First process
    waitingTime[0] = 0;
    completionTime[0] = burstTime[0];

    // Remaining processes
    for (int i = 1; i < totalProcesses; i++) {
        waitingTime[i] = completionTime[i - 1];
        completionTime[i] = waitingTime[i] + burstTime[i];
    }

    // Display Results
    double totalWaitingTime = 0;

    cout << "\nProcess\tBurst Time\tCompletion Time\tWaiting Time\n";

    for (int i = 0; i < totalProcesses; i++) {
        cout << processID[i] << "\t"
             << burstTime[i] << "\t\t"
             << completionTime[i] << "\t\t"
             << waitingTime[i] << endl;

        totalWaitingTime += waitingTime[i];
    }

    cout << "\nAverage Waiting Time: "
         << totalWaitingTime / totalProcesses << endl;

    return 0;
}
```

6. Input/Output



```
Enter number of processes: 4
Enter Burst Time for Process 1: 21
Enter Burst Time for Process 2: 3
Enter Burst Time for Process 3: 6
Enter Burst Time for Process 4: 2

Process Burst Time      Completion Time Waiting Time
4       2               2               0
2       3               5               2
3       6               11              5
1       21              32              11

Average Waiting Time: 4.5

Process returned 0 (0x0)   execution time : 5.523 s
Press any key to continue.
```

7. Conclusion

The SJF scheduling algorithm improves performance by executing shorter processes first, resulting in lower average waiting time compared to FCFS. However, it may cause starvation for longer processes and requires estimation of burst time.