Mohammed Nizamuddin
2503B05144
MTech CSE

Q1. A process is a program in execution. In real CPU scheduling, processes do **not always arrive at the same time**.
The First Come First Serve (FCFS) algorithm schedules processes in order of **arrival time**.

Requirements:

1. The program should accept:
    1. Number of processes
    2. Arrival Time (AT)
    3. Burst Time (BT)
2. The program must **sort the processes in ascending order of arrival time**.
3. The CPU may become **idle** if the next process arrives later. Your logic must correctly handle idle time.
4. Calculate for each process:
    1. Completion Time (CT)
    2. Turnaround Time (TAT = CT – AT)
    3. Waiting Time (WT = TAT – BT)
5. Display the result in a formatted table and print:
    1. Total TAT and WT
    2. Average TAT and WT
6. Use **non-preemptive FCFS**.

Given System State

Consider the following processes:

Process	Arrival Time	CPU Burst Time
P1	2	4

P2	1	3
P3	0	1
P4	3	2

Code:

```
#include <iostream>
using namespace std;

struct Process {
    int pid;       // Process ID
    int at;        // Arrival Time
    int bt;        // Burst Time
    int ct;        // Completion Time
    int tat;       // Turnaround Time
    int wt;        // Waiting Time
};

// Function to sort processes by arrival time
void sortByArrival(Process p[], int n) {
    for(int i = 0; i < n-1; i++) {
        for(int j = i+1; j < n; j++) {
            if(p[j].at < p[i].at) {
                Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

int main() {
    int n;
```

```
cout << "Enter number of processes: ";
cin >> n;

Process p[n];

// Input
for(int i = 0; i < n; i++) {
    p[i].pid = i + 1;
    cout << "Enter Arrival Time (AT) for P" << p[i].pid << ": ";
    cin >> p[i].at;
    cout << "Enter Burst Time (BT) for P" << p[i].pid << ": ";
    cin >> p[i].bt;
}

// Sort processes by Arrival Time
sortByArrival(p, n);

int currentTime = 0;
int totalTAT = 0, totalWT = 0;

// Calculate CT, TAT, WT
for(int i = 0; i < n; i++) {
    if(currentTime < p[i].at) {
        currentTime = p[i].at;   // CPU idle
    }

    p[i].ct = currentTime + p[i].bt;
    currentTime = p[i].ct;

    p[i].tat = p[i].ct - p[i].at;
    p[i].wt = p[i].tat - p[i].bt;

    totalTAT += p[i].tat;
    totalWT += p[i].wt;
}

// Output
```

```
    cout << "\nFCFS Scheduling Result:\n";
    cout << "PID\tAT\tBT\tCT\tTAT\tWT\n";

    for(int i = 0; i < n; i++) {
        cout << "P" << p[i].pid << "\t"
             << p[i].at << "\t"
             << p[i].bt << "\t"
             << p[i].ct << "\t"
             << p[i].tat << "\t"
             << p[i].wt << "\n";
    }

    cout << "\nTotal TAT = " << totalTAT;
    cout << "\nTotal WT = " << totalWT;
    cout << "\nAverage TAT = " << (float)totalTAT / n;
    cout << "\nAverage WT = " << (float)totalWT / n;

    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter Arrival Time (AT) for P1: 2
Enter Burst Time (BT) for P1: 4
Enter Arrival Time (AT) for P2: 1
Enter Burst Time (BT) for P2: 3
Enter Arrival Time (AT) for P3: 0
Enter Burst Time (BT) for P3: 1
Enter Arrival Time (AT) for P4: 3
Enter Burst Time (BT) for P4: 2

FCFS Scheduling Result:
PID AT  BT  CT  TAT WT
P3  0   1   1   1   0
P2  1   3   4   3   0
P1  2   4   8   6   2
P4  3   2   10  7   5

Total TAT = 17
Total WT = 7
Average TAT = 4.25
Average WT = 1.75

=== Code Execution Successful ===
```

2. **Assignment Question: Implement SJF (Shortest Job First) Scheduling Algorithm**

A process is a program in execution. CPU scheduling determines which process gets the CPU next.
The **Shortest Job First (SJF)** algorithm selects the process with the **smallest CPU burst time** next.

SJF can be:

- **Non-preemptive** → process runs until completion
- **Preemptive (SRTF)** → process may be interrupted if a new shorter job arrives

In this assignment, consider **Non-preemptive SJF**.

Problem Statement

Given the following processes:

Process	Arrival Time	CPU Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Task:
Implement the **Non-Preemptive SJF Scheduling Algorithm** to calculate:

1. Completion Time (CT)
2. Waiting Time (WT)
3. Turnaround Time (TAT)
4. Average Waiting Time
5. Average Turnaround Time
6. Gantt Chart

Code:

```
#include <iostream>
using namespace std;

struct Process {
    int pid;
```

```
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    bool completed;
};

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

    Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        cout << "Enter Arrival Time (AT) for P" << p[i].pid << ": ";
        cin >> p[i].at;
        cout << "Enter Burst Time (BT) for P" << p[i].pid << ": ";
        cin >> p[i].bt;
        p[i].completed = false;
    }

    int completed = 0, currentTime = 0;
    int totalTAT = 0, totalWT = 0;

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

```
    int minBT = 1e9;

    for (int i = 0; i < n; i++) {
        if (!p[i].completed && p[i].at <= currentTime && p[i].bt < minBT) {
            minBT = p[i].bt;
            idx = i;
        }
    }

    if (idx == -1) {
        currentTime++;   // CPU idle
        continue;
    }

    currentTime += p[idx].bt;
    p[idx].ct = currentTime;
    p[idx].tat = p[idx].ct - p[idx].at;
    p[idx].wt = p[idx].tat - p[idx].bt;

    p[idx].completed = true;
    completed++;

    totalTAT += p[idx].tat;
    totalWT += p[idx].wt;
}

cout << "\nNon-Preemptive SJF Scheduling Result:\n";
cout << "PID\tAT\tBT\tCT\tTAT\tWT\n";
for (int i = 0; i < n; i++) {
```

```
        cout << "P" << p[i].pid << "\t"
             << p[i].at << "\t"
             << p[i].bt << "\t"
             << p[i].ct << "\t"
             << p[i].tat << "\t"
             << p[i].wt << "\n";
    }

    cout << "\nTotal TAT = " << totalTAT;
    cout << "\nTotal WT = " << totalWT;
    cout << "\nAverage TAT = " << (float)totalTAT / n;
    cout << "\nAverage WT = " << (float)totalWT / n;

    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter Arrival Time (AT) for P1: 0
Enter Burst Time (BT) for P1: 8
Enter Arrival Time (AT) for P2: 1
Enter Burst Time (BT) for P2: 4
Enter Arrival Time (AT) for P3: 2
Enter Burst Time (BT) for P3: 9
Enter Arrival Time (AT) for P4: 3
Enter Burst Time (BT) for P4: 5

Non-Preemptive SJF Scheduling Result:
PID AT  BT  CT  TAT WT
P1  0   8   8   8   0
P2  1   4   12  11  7
P3  2   9   26  24  15
P4  3   5   17  14  9

Total TAT = 57
Total WT = 31
Average TAT = 14.25
Average WT = 7.75

=== Code Execution Successful ===
```