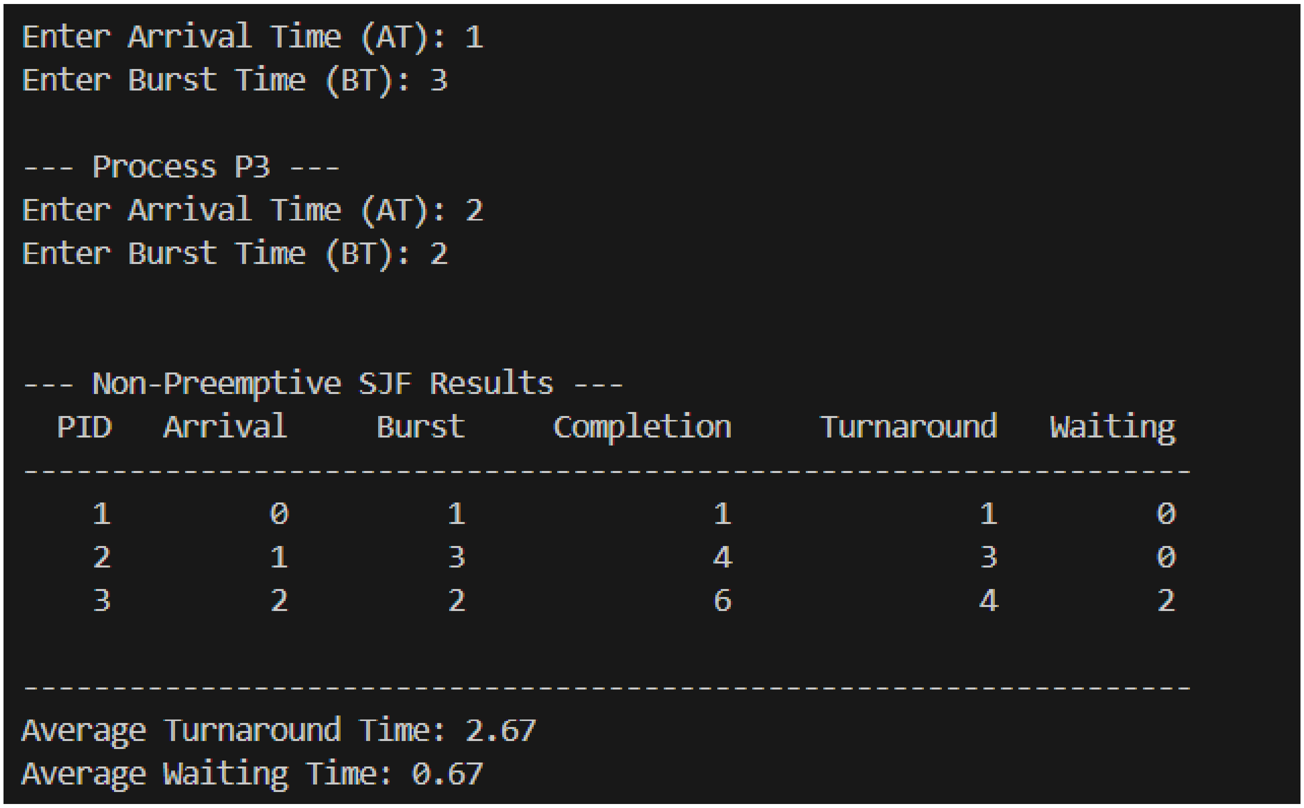
Sameer Gupta
OS Lab
November 9, 2025

## Practical 7

**Question 1.** Write a CPP program to simulate the CPU scheduling algorithm SJF(Shortest Job First).

a)Non Preemptive

```
#include <bits/stdc++.h>
using namespace std;
struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int completion_time;
    int waiting_time;
    int turnaround_time;
    int start_time;
    bool is_completed = false;
};
void get_user_input(vector<Process>& processes) {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
        Process p;
        p.id = i + 1;
        cout << "\n--- Process P" << p.id << " ---" << endl;
        cout << "Enter Arrival Time (AT): ";
        cin >> p.arrival_time;
        cout << "Enter Burst Time (BT): ";
        cin >> p.burst_time;
        p.remaining_time = p.burst_time;
        p.completion_time = 0;
        p.waiting_time = 0;
        p.turnaround_time = 0;
        p.start_time = 0;
        processes.push_back(p);
    }
}
void calculate_sjf_non_preemptive(vector<Process>& processes) {
    int n = processes.size();
    int current_time = 0;
    int completed_processes = 0;
    sort(processes.begin(), processes.end(), [](const Process& a, const Process& b) {
        return a.arrival_time < b.arrival_time;
    });
    while (completed_processes != n) {
        int shortest_job_index = -1;
        int min_burst_time = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && !processes[i].is_completed) {
                if (processes[i].burst_time < min_burst_time) {
                    min_burst_time = processes[i].burst_time;
                    shortest_job_index = i;
                }
            }
        }
        if (shortest_job_index != -1) {
            int index = shortest_job_index;
            processes[index].start_time = current_time;
            processes[index].completion_time = processes[index].start_time + processes[index].burst_
            current_time = processes[index].completion_time;
            processes[index].is_completed = true;
            completed_processes++;
            processes[index].turnaround_time =
                processes[index].completion_time - processes[index].arrival_time;
            processes[index].waiting_time =
                processes[index].turnaround_time - processes[index].burst_time;
        } else {
            current_time++;
        }
    }
    double total_waiting_time = 0;
    double total_turnaround_time = 0;
    cout << "\n\n--- Non-Preemptive SJF Results ---" << endl;
    cout << setw(5) << "PID" << setw(10) << "Arrival" << setw(10) << "Burst"
         << setw(15) << "Completion" << setw(15) << "Turnaround" << setw(10) << "Waiting" << endl;
    cout << "------------------------------------------------------------------" << endl;
    for (const auto& p : processes) {
        total_waiting_time += p.waiting_time;
        total_turnaround_time += p.turnaround_time;
        cout << setw(5) << p.id
             << setw(10) << p.arrival_time
             << setw(10) << p.burst_time
             << setw(15) << p.completion_time
             << setw(15) << p.turnaround_time
             << setw(10) << p.waiting_time << endl;
    }
    cout << "\n------------------------------------------------------------------" << endl;
    cout << fixed << setprecision(2);
    cout << "Average Turnaround Time: " << total_turnaround_time / n << endl;
    cout << "Average Waiting Time: " << total_waiting_time / n << endl;
}
int main() {
    vector<Process> processes;
    get_user_input(processes);
    if (!processes.empty()) {
        calculate_sjf_non_preemptive(processes);
    } else {
        cout << "No processes entered. Exiting." << endl;
    }
    return 0;
}
```

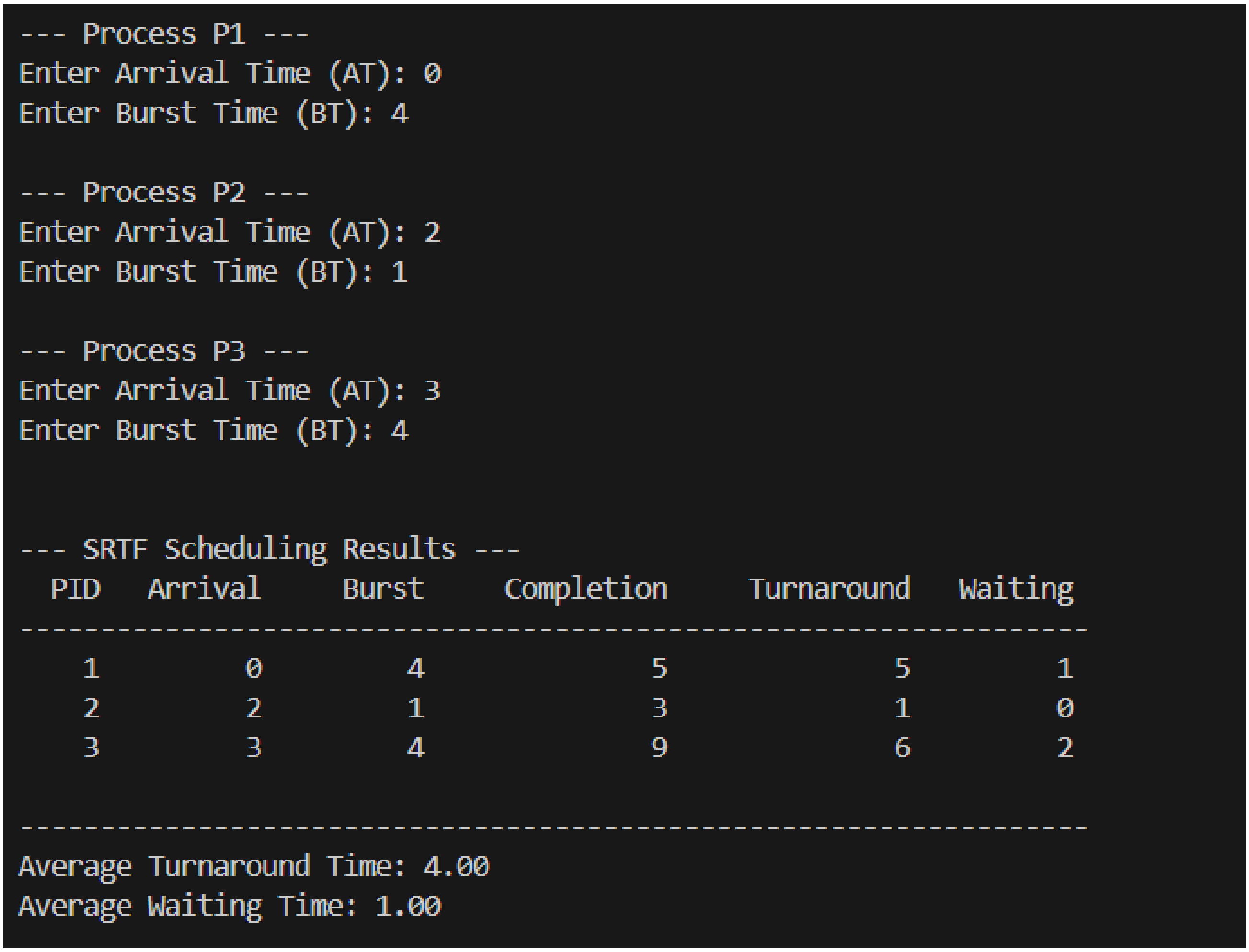


```
Enter Arrival Time (AT): 1
Enter Burst Time (BT): 3

--- Process P3 ---
Enter Arrival Time (AT): 2
Enter Burst Time (BT): 2


--- Non-Preemptive SJF Results ---
  PID   Arrival     Burst     Completion     Turnaround   Waiting
------------------------------------------------------------------
    1         0         1              1              1         0
    2         1         3              4              3         0
    3         2         2              6              4         2

------------------------------------------------------------------
Average Turnaround Time: 2.67
Average Waiting Time: 0.67
```

FIGURE 1. SJF Non-preemptive example

b)Preemptive

```
#include <bits/stdc++.h>
using namespace std;
struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int completion_time;
    int waiting_time;
    int turnaround_time;
    int start_time;
};
void get_user_input(vector<Process>& processes) {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
        Process p;
        p.id = i + 1;
        cout << "\n--- Process P" << p.id << " ---" << endl;
        cout << "Enter Arrival Time (AT): ";
        cin >> p.arrival_time;
        cout << "Enter Burst Time (BT): ";
        cin >> p.burst_time;
        p.remaining_time = p.burst_time;
        p.completion_time = 0;
        p.waiting_time = 0;
        p.turnaround_time = 0;
        p.start_time = 0;
        processes.push_back(p);
    }
}
void calculate_srtf(vector<Process>& processes) {
    int n = processes.size();
    int current_time = 0;
    int completed_processes = 0;
    int shortest_job_index = -1;
    int min_remaining_time = INT_MAX;
    int total_burst_time = 0;
    int max_arrival_time = 0;
    for (const auto& p : processes) {
        total_burst_time += p.burst_time;
        max_arrival_time = max(max_arrival_time, p.arrival_time);
    }
    int max_possible_time = total_burst_time + max_arrival_time;
    bool process_found = false;
    while (completed_processes != n) {
        min_remaining_time = INT_MAX;
        shortest_job_index = -1;
        process_found = false;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && processes[i].remaining_time > 0) {
                if (processes[i].remaining_time < min_remaining_time) {
                    min_remaining_time = processes[i].remaining_time;
                    shortest_job_index = i;
                    process_found = true;
                }
            }
        }
        if (process_found) {
            processes[shortest_job_index].remaining_time--;
            current_time++;
            if (processes[shortest_job_index].remaining_time == 0) {
                completed_processes++;
                processes[shortest_job_index].completion_time = current_time;
                processes[shortest_job_index].turnaround_time =
                    processes[shortest_job_index].completion_time - processes[shortest_job_index].ar
                processes[shortest_job_index].waiting_time =
                    processes[shortest_job_index].turnaround_time - processes[shortest_job_index].bu
            }
        } else {
            current_time++;
        }
        if (current_time > max_possible_time + n) break;
    }
    double total_waiting_time = 0;
    double total_turnaround_time = 0;
    cout << "\n\n--- SRTF Scheduling Results ---" << endl;
    cout << setw(5) << "PID" << setw(10) << "Arrival" << setw(10) << "Burst"
         << setw(15) << "Completion" << setw(15) << "Turnaround" << setw(10) << "Waiting" << endl;
    cout << "--------------------------------------------------------------------" << endl;
    for (const auto& p : processes) {
        total_waiting_time += p.waiting_time;
        total_turnaround_time += p.turnaround_time;
        cout << setw(5) << p.id
             << setw(10) << p.arrival_time
             << setw(10) << p.burst_time
             << setw(15) << p.completion_time
             << setw(15) << p.turnaround_time
             << setw(10) << p.waiting_time << endl;
    }
    cout << "\n------------------------------------------------------------------" << endl;
    cout << fixed << setprecision(2);
    cout << "Average Turnaround Time: " << total_turnaround_time / n << endl;
    cout << "Average Waiting Time: " << total_waiting_time / n << endl;
}
int main() {
    vector<Process> processes;
    get_user_input(processes);
    if (!processes.empty()) {
        calculate_srtf(processes);
    } else {
        cout << "No processes entered. Exiting." << endl;
    }
    return 0;
}
```



```
--- Process P1 ---
Enter Arrival Time (AT): 0
Enter Burst Time (BT): 4

--- Process P2 ---
Enter Arrival Time (AT): 2
Enter Burst Time (BT): 1

--- Process P3 ---
Enter Arrival Time (AT): 3
Enter Burst Time (BT): 4


--- SRTF Scheduling Results ---
  PID   Arrival     Burst     Completion     Turnaround   Waiting
------------------------------------------------------------------
    1         0         4              5              5         1
    2         2         1              3              1         0
    3         3         4              9              6         2

------------------------------------------------------------------
Average Turnaround Time: 4.00
Average Waiting Time: 1.00
```

FIGURE 2. SJF preemptive example