

Practical 7

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

a) Non Preemptive

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct Process {
4     int id;
5     int arrival_time;
6     int burst_time;
7     int remaining_time;
8     int completion_time;
9     int waiting_time;
10    int turnaround_time;
11    int start_time;
12    bool is_completed = false;
13};
14 void get_user_input(vector<Process>& processes) {
15     int n;
16     cout << "Enter the number of processes: ";
17     cin >> n;
18     for (int i = 0; i < n; ++i) {
19         Process p;
20         p.id = i + 1;
21         cout << "\n--- Process P" << p.id << " ---" << endl;
22         cout << "Enter Arrival Time (AT): ";
23         cin >> p.arrival_time;
24         cout << "Enter Burst Time (BT): ";
25         cin >> p.burst_time;
26         p.remaining_time = p.burst_time;
27         p.completion_time = 0;
28         p.waiting_time = 0;
29         p.turnaround_time = 0;
30         p.start_time = 0;
31         processes.push_back(p);
32     }
33 }
34 void calculate_sjf_non_preemptive(vector<Process>& processes) {
35     int n = processes.size();
36     int current_time = 0;
37     int completed_processes = 0;
38     sort(processes.begin(), processes.end(), [] (const Process& a, const Process& b) {
39         return a.arrival_time < b.arrival_time;
40     });
41     while (completed_processes != n) {
42         int shortest_job_index = -1;
43         int min_burst_time = INT_MAX;
44         for (int i = 0; i < n; i++) {
45             if (processes[i].arrival_time <= current_time && !processes[i].is_completed) {
46                 if (processes[i].burst_time < min_burst_time) {
47                     min_burst_time = processes[i].burst_time;
48                     shortest_job_index = i;
49                 }
50             }
51         }
52         if (shortest_job_index != -1) {
53             processes[shortest_job_index].start_time = current_time;
54             current_time += processes[shortest_job_index].burst_time;
55             processes[shortest_job_index].completion_time = current_time;
56             processes[shortest_job_index].waiting_time = current_time - processes[shortest_job_index].arrival_time;
57             processes[shortest_job_index].turnaround_time = processes[shortest_job_index].completion_time - processes[shortest_job_index].arrival_time;
58             processes[shortest_job_index].is_completed = true;
59             completed_processes++;
60         }
61     }
62 }
```

```

50     }
51 }
52 if (shortest_job_index != -1) {
53     int index = shortest_job_index;
54     processes[index].start_time = current_time;
55     processes[index].completion_time = processes[index].start_time + processes[index].burst_
56     current_time = processes[index].completion_time;
57     processes[index].is_completed = true;
58     completed_processes++;
59     processes[index].turnaround_time =
60         processes[index].completion_time - processes[index].arrival_time;
61     processes[index].waiting_time =
62         processes[index].turnaround_time - processes[index].burst_time;
63 } else {
64     current_time++;
65 }
66 }
67 double total_waiting_time = 0;
68 double total_turnaround_time = 0;
69 cout << "\n\n--- Non-Preemptive SJF Results ---" << endl;
70 cout << setw(5) << "PID" << setw(10) << "Arrival" << setw(10) << "Burst"
71     << setw(15) << "Completion" << setw(15) << "Turnaround" << setw(10) << "Waiting" << endl;
72 cout << "-----" << endl;
73 for (const auto& p : processes) {
74     total_waiting_time += p.waiting_time;
75     total_turnaround_time += p.turnaround_time;
76     cout << setw(5) << p.id
77         << setw(10) << p.arrival_time
78         << setw(10) << p.burst_time
79         << setw(15) << p.completion_time
80         << setw(15) << p.turnaround_time
81         << setw(10) << p.waiting_time << endl;
82 }
83 cout << "\n-----" << endl;
84 cout << fixed << setprecision(2);
85 cout << "Average Turnaround Time: " << total_turnaround_time / n << endl;
86 cout << "Average Waiting Time: " << total_waiting_time / n << endl;
87 }
88 int main() {
89     vector<Process> processes;
90     get_user_input(processes);
91     if (!processes.empty()) {
92         calculate_sjf_non_preemptive(processes);
93     } else {
94         cout << "No processes entered. Exiting." << endl;
95     }
96     return 0;
97 }
```

```

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             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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct Process {
4     int id;
5     int arrival_time;
6     int burst_time;
7     int remaining_time;
8     int completion_time;
9     int waiting_time;
10    int turnaround_time;
11    int start_time;
12};
13 void get_user_input(vector<Process>& processes) {
14     int n;
15     cout << "Enter the number of processes: ";
16     cin >> n;
17     for (int i = 0; i < n; ++i) {
18         Process p;
19         p.id = i + 1;
20         cout << "\n--- Process P" << p.id << " ---" << endl;
21         cout << "Enter Arrival Time (AT): ";
22         cin >> p.arrival_time;
23         cout << "Enter Burst Time (BT): ";
24         cin >> p.burst_time;
25         p.remaining_time = p.burst_time;
26         p.completion_time = 0;
27         p.waiting_time = 0;

```

```

28     p.turnaround_time = 0;
29     p.start_time = 0;
30     processes.push_back(p);
31 }
32 }
33 void calculate_srtf(vector<Process>& processes) {
34     int n = processes.size();
35     int current_time = 0;
36     int completed_processes = 0;
37     int shortest_job_index = -1;
38     int min_remaining_time = INT_MAX;
39     int total_burst_time = 0;
40     int max_arrival_time = 0;
41     for (const auto& p : processes) {
42         total_burst_time += p.burst_time;
43         max_arrival_time = max(max_arrival_time, p.arrival_time);
44     }
45     int max_possible_time = total_burst_time + max_arrival_time;
46     bool process_found = false;
47     while (completed_processes != n) {
48         min_remaining_time = INT_MAX;
49         shortest_job_index = -1;
50         process_found = false;
51         for (int i = 0; i < n; i++) {
52             if (processes[i].arrival_time <= current_time && processes[i].remaining_time > 0) {
53                 if (processes[i].remaining_time < min_remaining_time) {
54                     min_remaining_time = processes[i].remaining_time;
55                     shortest_job_index = i;
56                     process_found = true;
57                 }
58             }
59         }
60         if (process_found) {
61             processes[shortest_job_index].remaining_time--;
62             current_time++;
63             if (processes[shortest_job_index].remaining_time == 0) {
64                 completed_processes++;
65                 processes[shortest_job_index].completion_time = current_time;
66                 processes[shortest_job_index].turnaround_time =
67                     processes[shortest_job_index].completion_time - processes[shortest_job_index].ar-
68                     processes[shortest_job_index].waiting_time =
69                     processes[shortest_job_index].turnaround_time - processes[shortest_job_index].bu-
70                 }
71             } else {
72                 current_time++;
73             }
74             if (current_time > max_possible_time + n) break;
75         }
76         double total_waiting_time = 0;
77         double total_turnaround_time = 0;
78         cout << "\n\n--- SRTF Scheduling Results ---" << endl;
79         cout << setw(5) << "PID" << setw(10) << "Arrival" << setw(10) << "Burst"
80             << setw(15) << "Completion" << setw(15) << "Turnaround" << setw(10) << "Waiting" << endl;
81         cout << "-----" << endl;
82         for (const auto& p : processes) {
83             total_waiting_time += p.waiting_time;

```

```

84     total_turnaround_time += p.turnaround_time;
85     cout << setw(5) << p.id
86         << setw(10) << p.arrival_time
87         << setw(10) << p.burst_time
88         << setw(15) << p.completion_time
89         << setw(15) << p.turnaround_time
90         << setw(10) << p.waiting_time << endl;
91 }
92 cout << "\n-----" << endl;
93 cout << fixed << setprecision(2);
94 cout << "Average Turnaround Time: " << total_turnaround_time / n << endl;
95 cout << "Average Waiting Time: " << total_waiting_time / n << endl;
96 }
97 int main() {
98     vector<Process> processes;
99     get_user_input(processes);
100    if (!processes.empty()) {
101        calculate_srtf(processes);
102    } else {
103        cout << "No processes entered. Exiting." << endl;
104    }
105    return 0;
106 }
```

```

--- 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