

## Scheduling Algorithm

### 1. Round Robin Algorithm Implementation code:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #define MAX_PROCESSES 10
4  struct Process {
5      char name[10];
6      int burst_time;
7      int remaining_time;
8  };
9  void round_robin(struct Process processes[], int n, int quantum) {
10     struct Process* queue[MAX_PROCESSES];
11     int front = 0, rear = 0;
12     int time = 0;
13     int total_waiting_time = 0;
14
15     for (int i = 0; i < n; i++) {
16         queue[rear++] = &processes[i];
17     }
18     while (front != rear) {
19         struct Process* current_process = queue[front++];
20         if (current_process->remaining_time > quantum) {
21             time += quantum;
22             current_process->remaining_time -= quantum;
23             queue[rear++] = current_process;
24         } else {
25             time += current_process->remaining_time;
26             current_process->remaining_time = 0;
27             int waiting_time = time - current_process->burst_time;
28             total_waiting_time += waiting_time;
29             printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time);
30         }
31     }
32     float average_waiting_time = (float)total_waiting_time / n;
33     printf("\nAverage waiting time: %.2f\n", average_waiting_time);
34 }
35 int main() {
36     struct Process processes[MAX_PROCESSES] = {
37         {"P1", 10, 0},
38         {"P2", 5, 0},
39         {"P3", 8, 0},
40         {"P4", 12, 0}
41     };
42     int n = 4;
43     int quantum = 4;
44     round_robin(processes, n, quantum);
45     return 0;
46 }
```

Output:

```
Process P1 completed. Waiting time: -10
Process P2 completed. Waiting time: -5
Process P3 completed. Waiting time: -8
Process P4 completed. Waiting time: -12

Average waiting time: -8.75
```

## 2. Shortest Job First:

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_PROCESSES 10
4 struct Process {
5     char name[10];
6     int arrival_time;
7     int burst_time;
8     int remaining_time;
9     bool completed;
10 };
11 void shortest_job_first(struct Process processes[], int n) {
12     int time = 0;
13     int total_waiting_time = 0;
14     while (true) {
15         int shortest_job_index = -1;
16         int shortest_burst_time = -1;
17         // Find the process with the shortest remaining burst time
18         for (int i = 0; i < n; i++) {
19             if (!processes[i].completed && processes[i].arrival_time <= time) {
20                 if (shortest_job_index == -1 || processes[i].burst_time < shortest_burst_time) {
21                     shortest_job_index = i;
22                     shortest_burst_time = processes[i].burst_time;
23                 }
24             }
25         }
26         if (shortest_job_index == -1) {
27             // No eligible processes found, increment time
28             time++;
29             continue;
30         }
31         struct Process* current_process = &processes[shortest_job_index];
32         current_process->completed = true;
33         current_process->remaining_time = 0;
34         int waiting_time = time - current_process->arrival_time;
35         total_waiting_time += waiting_time;
36         printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time);
37         time += current_process->burst_time;
38         // Check if all processes have completed
39         bool all_completed = true;
40         for (int i = 0; i < n; i++) {
41             if (!processes[i].completed) {
42                 all_completed = false;
43                 break;
44             }
45         }
46         if (all_completed) {
47             break;
48         }
49     }
50     float average_waiting_time = (float)total_waiting_time / n;
51     printf("\nAverage waiting time: %.2f\n", average_waiting_time);
52 }
53 int main() {
54     struct Process processes[MAX_PROCESSES] = {
55         {"P1", 0, 10, 0, false},
56         {"P2", 2, 5, 0, false},
57         {"P3", 4, 8, 0, false},
58         {"P4", 5, 12, 0, false}
59     };
60     int n = 4;
61     shortest_job_first(processes, n);
62     return 0;
63 }
```

Output:

```
Process P1 completed. Waiting time: 0
Process P2 completed. Waiting time: 8
Process P3 completed. Waiting time: 11
Process P4 completed. Waiting time: 18

Average waiting time: 9.25
```

### 3. Shortest Remaining Time Next:

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_PROCESSES 10
4 struct Process {
5     char name[10];
6     int arrival_time;
7     int burst_time;
8     int remaining_time;
9     bool completed;
10 };
11 void shortest_remaining_time_next(struct Process processes[], int n) {
12     int time = 0;
13     int total_waiting_time = 0;
14     int completed_processes = 0;
15     while (completed_processes < n) {
16         int shortest_job_index = -1;
17         int shortest_burst_time = -1;
18         // Find the process with the shortest remaining burst time
19         for (int i = 0; i < n; i++) {
20             if (!processes[i].completed && processes[i].arrival_time <= time) {
21                 if (shortest_job_index == -1 || processes[i].remaining_time < shortest_burst_time) {
22                     shortest_job_index = i;
23                     shortest_burst_time = processes[i].remaining_time;
24                 }
25             }
26         }
27         if (shortest_job_index == -1) {
28             // No eligible processes found, increment time
29             time++;
30             continue;
31         }
32         struct Process* current_process = &processes[shortest_job_index];
33         current_process->remaining_time--;
34         current_process->completed = (current_process->remaining_time == 0);
35         if (current_process->completed) {
36             completed_processes++;
37             int waiting_time = time + 1 - current_process->arrival_time - current_process->burst_time;
38             total_waiting_time += waiting_time;
39             printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time);
40         }
41         time++;
42     }
43     float average_waiting_time = (float)total_waiting_time / n;
44     printf("\nAverage waiting time: %.2f\n", average_waiting_time);
45 }
46 int main() {
47     struct Process processes[MAX_PROCESSES] = {
48         {"P1", 0, 10, 10, false},
49         {"P2", 2, 5, 5, false},
50         {"P3", 4, 8, 8, false},
51         {"P4", 5, 12, 12, false}
52     };
53     int n = 4;
54     shortest_remaining_time_next(processes, n);
55     return 0;
56 }
```

Output:

```
Process P2 completed. Waiting time: 0
Process P1 completed. Waiting time: 5
Process P3 completed. Waiting time: 11
Process P4 completed. Waiting time: 18

Average waiting time: 8.50
```

#### 4. First in First Out:

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_PROCESSES 10
4 struct Process {
5     char name[10];
6     int burst_time;
7     int waiting_time;
8     int turnaround_time;
9 };
10 void first_in_first_out(struct Process processes[], int n) {
11     int total_waiting_time = 0;
12     int total_turnaround_time = 0;
13     // Calculate waiting time and turnaround time for each process
14     for (int i = 0; i < n; i++) {
15         if (i > 0) {
16             processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
17         }
18         else {
19             processes[i].waiting_time = 0;
20         }
21         processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
22         total_waiting_time += processes[i].waiting_time;
23         total_turnaround_time += processes[i].turnaround_time;
24     }
25     float average_waiting_time = (float)total_waiting_time / n;
26     float average_turnaround_time = (float)total_turnaround_time / n;
27     // Display process details and average times
28     for (int i = 0; i < n; i++) {
29         printf("Process %s\n", processes[i].name);
30         printf("Burst Time: %d\n", processes[i].burst_time);
31         printf("Waiting Time: %d\n", processes[i].waiting_time);
32         printf("Turnaround Time: %d\n\n", processes[i].turnaround_time);
33     }
34     printf("Average Waiting Time: %.2f\n", average_waiting_time);
35     printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
36 }
37 int main() {
38     struct Process processes[MAX_PROCESSES] = {
39         {"P1", 10, 0, 0},
40         {"P2", 5, 0, 0},
41         {"P3", 8, 0, 0},
42         {"P4", 12, 0, 0}
43     };
44     int n = 4;
45     first_in_first_out(processes, n);
46     return 0;
47 }
```

Output:

```
Process P1
Burst Time: 10
Waiting Time: 0
Turnaround Time: 10
```

```
Process P2
Burst Time: 5
Waiting Time: 10
Turnaround Time: 15
```

```
Process P3
Burst Time: 8
Waiting Time: 15
Turnaround Time: 23
```

```
Process P4
Burst Time: 12
Waiting Time: 23
Turnaround Time: 35
```

```
Average Waiting Time: 12.00
Average Turnaround Time: 20.75
```

## 5. Priority Scheduling:

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_PROCESSES 10
4 struct Process {
5     char name[10];
6     int burst_time;
7     int priority;
8     int waiting_time;
9     int turnaround_time;
10 };
11 void priority_scheduling(struct Process processes[], int n) {
12     int total_waiting_time = 0;
13     int total_turnaround_time = 0;
14     // Sort the processes based on priority (ascending order)
15     for (int i = 0; i < n - 1; i++) {
16         for (int j = 0; j < n - i - 1; j++) {
17             if (processes[j].priority > processes[j + 1].priority) {
18                 // Swap processes
19                 struct Process temp = processes[j];
20                 processes[j] = processes[j + 1];
21                 processes[j + 1] = temp;
22             }
23         }
24     }
25     // Calculate waiting time and turnaround time for each process
26     for (int i = 0; i < n; i++) {
27         if (i > 0) {
28             processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
29         }
30         else {
31             processes[i].waiting_time = 0;
32         }
33         processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
34         total_waiting_time += processes[i].waiting_time;
35         total_turnaround_time += processes[i].turnaround_time;
36     }
37     float average_waiting_time = (float)total_waiting_time / n;
38     float average_turnaround_time = (float)total_turnaround_time / n;
39     // Display process details and average times
40     for (int i = 0; i < n; i++) {
41         printf("Process %s\n", processes[i].name);
42         printf("Burst Time: %d\n", processes[i].burst_time);
43         printf("Priority: %d\n", processes[i].priority);
44         printf("Waiting Time: %d\n", processes[i].waiting_time);
45         printf("Turnaround Time: %d\n\n", processes[i].turnaround_time);
46     }
47     printf("Average Waiting Time: %.2f\n", average_waiting_time);
48     printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
49 }
50 int main() {
51     struct Process processes[MAX_PROCESSES] = {
52         {"P1", 10, 2, 0, 0},
53         {"P2", 5, 1, 0, 0},
54         {"P3", 8, 4, 0, 0},
55         {"P4", 12, 3, 0, 0}
56     };
57     int n = 4;
58     priority_scheduling(processes, n);
59     return 0;
60 }
```

Output:

```
Process P2
Burst Time: 5
Priority: 1
Waiting Time: 0
Turnaround Time: 5

Process P1
Burst Time: 10
Priority: 2
Waiting Time: 5
Turnaround Time: 15

Process P4
Burst Time: 12
Priority: 3
Waiting Time: 15
Turnaround Time: 27

Process P3
Burst Time: 8
Priority: 4
Waiting Time: 27
Turnaround Time: 35

Average Waiting Time: 11.75
Average Turnaround Time: 20.50
```