Scheduling Algorithm

1. Round Robin Algorithm Implementation code:

```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
struct Process {
    char name[10];
    int burst_time;
    int remaining_time;
};
void round_robin(struct Process processes[], int n, int quantum) {
    struct Process* queue[MAX_PROCESSES];
    int front = 0, rear = 0;
    int time = 0;
    int total_waiting_time = 0;

    for (int i = 0; i < n; i++) {
        queue[rear++] = &processes[i];
    }
    while (front != rear) {
        struct Process* current_process = queue[front++];
        if (current_process->remaining_time > quantum) {
            time += quantum;
            current_process->remaining_time -= quantum;
            queue[rear++] = current_process;
        } else {
            time += current_process->remaining_time;
            current_process->remaining_time = 0;
            int waiting_time = time - current_process->burst_time;
            total_waiting_time += waiting_time;
            printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time
        }
    }float average_waiting_time = (float)total_waiting_time / n;
    printf("\nAverage waiting time: %.2f\n", average_waiting_time);
}
int main() {
    struct Process processes[MAX_PROCESSES] = {
        {"P1", 10, 0},
        {"P2", 5, 0},
        {"P3", 8, 0},
        {"P4", 12, 0}
    };
    int n = 4;
    int quantum = 4;
    round_robin(processes, n, quantum);
    return 0;
}
```

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:

```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
struct Process {
    char name[10];
    int arrival_time;
    int burst_time;
    int remaining_time;
    bool completed;
};
void shortest_job_first(struct Process processes[], int n) {
    int time = 0;
    int total_waiting_time = 0;
    while (true) {
        int shortest_job_index = -1;
        int shortest_burst_time = -1;
        // Find the process with the shortest remaining burst time
        for (int i = 0; i < n; i++) {
            if (!processes[i].completed && processes[i].arrival_time <= time) {
                if (shortest_job_index == -1 || processes[i].burst_time < shortest_burst_time) {
                    shortest_job_index = i;
                    shortest_burst_time = processes[i].burst_time;
                }
            }
        }
        if (shortest_job_index == -1) {
            // No eligible processes found, increment time
            time++;
            continue;
        }
        struct Process* current_process = &processes[shortest_job_index];
        current_process->completed = true;
        current_process->remaining_time = 0;
        int waiting_time = time - current_process->arrival_time;
        total_waiting_time += waiting_time;
        printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time);
        time += current_process->burst_time;
        // Check if all processes have completed
        bool all_completed = true;
        for (int i = 0; i < n; i++) {
            if (!processes[i].completed) {
                all_completed = false;
                break;
            }
        }
        if (all_completed) {
            break;
        }
    }
    float average_waiting_time = (float)total_waiting_time / n;
    printf("\nAverage waiting time: %.2f\n", average_waiting_time);
}
int main() {
    struct Process processes[MAX_PROCESSES] = {
        {"P1", 0, 10, 0, false},
        {"P2", 2, 5, 0, false},
        {"P3", 4, 8, 0, false},
        {"P4", 5, 12, 0, false}
    };
    int n = 4;
    shortest_job_first(processes, n);
    return 0;
}
```

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:

```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
struct Process {
    char name[10];
    int arrival_time;
    int burst_time;
    int remaining_time;
    bool completed;
};
void shortest_remaining_time_next(struct Process processes[], int n) {
    int time = 0;
    int total_waiting_time = 0;
    int completed_processes = 0;
    while (completed_processes < n) {
        int shortest_job_index = -1;
        int shortest_burst_time = -1;
        // Find the process with the shortest remaining burst time
        for (int i = 0; i < n; i++) {
            if (!processes[i].completed && processes[i].arrival_time <= time) {
                if (shortest_job_index == -1 || processes[i].remaining_time < shortest_burst_time
                    shortest_job_index = i;
                    shortest_burst_time = processes[i].remaining_time;
                }
            }
        }
        if (shortest_job_index == -1) {
            // No eligible processes found, increment time
            time++;
            continue;
        }
        struct Process* current_process = &processes[shortest_job_index];
        current_process->remaining_time--;
        current_process->completed = (current_process->remaining_time == 0);
       if (current_process->completed) {
            completed_processes++;
            int waiting_time = time + 1 - current_process->arrival_time - current_process->burst_time;
            total_waiting_time += waiting_time;
            printf("Process %s completed. Waiting time: %d\n", current_process->name, waiting_time);
        }
        time++;
    }
    float average_waiting_time = (float)total_waiting_time / n;
    printf("\nAverage waiting time: %.2f\n", average_waiting_time);
}
int main() {
    struct Process processes[MAX_PROCESSES] = {
        {"P1", 0, 10, 10, false},
        {"P2", 2, 5, 5, false},
        {"P3", 4, 8, 8, false},
        {"P4", 5, 12, 12, false}
    };
    int n = 4;
    shortest_remaining_time_next(processes, n);
    return 0;
}
```

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:

```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
struct Process {
    char name[10];
    int burst_time;
    int waiting_time;
    int turnaround_time;
};
void first_in_first_out(struct Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    // Calculate waiting time and turnaround time for each process
    for (int i = 0; i < n; i++) {
        if (i > 0) {
            processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        }
        else {
            processes[i].waiting_time = 0;
        }
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
    float average_waiting_time = (float)total_waiting_time / n;
    float average_turnaround_time = (float)total_turnaround_time / n;
    // Display process details and average times
    for (int i = 0; i < n; i++) {
        printf("Process %s\n", processes[i].name);
        printf("Burst Time: %d\n", processes[i].burst_time);
        printf("Waiting Time: %d\n", processes[i].waiting_time);
        printf("Turnaround Time: %d\n\n", processes[i].turnaround_time);
    }
    printf("Average Waiting Time: %.2f\n", average_waiting_time);
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
}
int main() {
    struct Process processes[MAX_PROCESSES] = {
        {"P1", 10, 0, 0},
        {"P2", 5, 0, 0},
        {"P3", 8, 0, 0},
        {"P4", 12, 0, 0}
    };
    int n = 4;
    first_in_first_out(processes, n);
    return 0;
}
```

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:

```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
struct Process {
    char name[10];
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};
void priority_scheduling(struct Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    // Sort the processes based on priority (ascending order)
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].priority > processes[j + 1].priority) {
                // Swap processes
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
    // Calculate waiting time and turnaround time for each process
    for (int i = 0; i < n; i++) {
        if (i > 0) {
            processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        }
        else {
            processes[i].waiting_time = 0;
        }
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
         total_waiting_time += processes[i].waiting_time;
         total_turnaround_time += processes[i].turnaround_time;
    }
    float average_waiting_time = (float)total_waiting_time / n;
    float average_turnaround_time = (float)total_turnaround_time / n;
    // Display process details and average times
    for (int i = 0; i < n; i++) {
        printf("Process %s\n", processes[i].name);
        printf("Burst Time: %d\n", processes[i].burst_time);
        printf("Priority: %d\n", processes[i].priority);
        printf("Waiting Time: %d\n", processes[i].waiting_time);
        printf("Turnaround Time: %d\n\n", processes[i].turnaround_time);
    }
    printf("Average Waiting Time: %.2f\n", average_waiting_time);
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
}
int main() {
    struct Process processes[MAX_PROCESSES] = {
        {"P1", 10, 2, 0, 0},
        {"P2", 5, 1, 0, 0},
        {"P3", 8, 4, 0, 0},
        {"P4", 12, 3, 0, 0}
    };
    int n = 4;
    priority_scheduling(processes, n);
    return 0;
}
```

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
```