Code

FCFS

```
#include <stdio.h>
struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};
void calculateWaitingTime(struct Process processes[], int n) {
    int total_waiting_time = 0;
    int i;
    processes[0].waiting_time = 0;
    for (i = 1; i < n; i++) {
        processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        total_waiting_time += processes[i].waiting_time;
    }
    double avg_waiting_time = (double)total_waiting_time / n;
    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
}
void calculateTurnaroundTime(struct Process processes[], int n) {
    int total_turnaround_time = 0;
    int i;
    for (i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
    double avg_turnaround_time = (double)total_turnaround_time / n;
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}
void scheduleFCFS(struct Process processes[], int n) {
```

```
    printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].process_id, processes[i].arrival_time,
               processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\n");
    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);
}
int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].process_id = i + 1;
    }
    scheduleFCFS(processes, n);}
```

SJF

```
#include <stdio.h>
struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
```

```
};
void calculateWaitingTime(struct Process processes[], int n) {
    int remaining_time[n];
    int completed = 0;
    int current_time = 0;
    int shortest_job = 0;
    int i;
    for (i = 0; i < n; i++) {
        remaining_time[i] = processes[i].burst_time;
    }
    while (completed != n) {
        shortest_job = -1;
        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && remaining_time[i] > 0) {
                if (shortest_job == -1 || remaining_time[i] < remaining_time[shortest_job]) {
                    shortest_job = i;
                }
            }
        }
        if (shortest_job == -1) {
            current_time++;
        } else {
            processes[shortest_job].waiting_time = current_time - processes[shortest_job].arrival_time;
            current_time += processes[shortest_job].burst_time;
            processes[shortest_job].turnaround_time = current_time - processes[shortest_job].arrival_time;
            remaining_time[shortest_job] = 0;
            completed++;
        }
    }
    int total_waiting_time = 0;
    for (i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
```

```
    }
    double avg_waiting_time = (double)total_waiting_time / n;
    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
}
void calculateTurnaroundTime(struct Process processes[], int n) {
    int total_turnaround_time = 0;
    int i;
    for (i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
    double avg_turnaround_time = (double)total_turnaround_time / n;
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}
void scheduleSJF(struct Process processes[], int n) {
    printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].process_id, processes[i].arrival_time,
               processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\n");
    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);
}
int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
```

```
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].process_id = i + 1;
    }
    scheduleSJF(processes, n);
    return 0;
}
```

Round Robin

```
#include <stdio.h>
#define MAX_PROCESSES 10
struct Process {
    int process_id;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
};
void calculateWaitingTime(struct Process processes[], int n, int quantum) {
    int i, completed = 0, current_time = 0;
    while (completed < n) {
        for (i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                if (processes[i].remaining_time > quantum) {
                    current_time += quantum;
                    processes[i].remaining_time -= quantum;
                } else {
                    current_time += processes[i].remaining_time;
                    processes[i].waiting_time = current_time - processes[i].burst_time;
```

```
                processes[i].remaining_time = 0;
                completed++;
            }
        }
    }
}
void calculateTurnaroundTime(struct Process processes[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].burst_time + processes[i].waiting_time;
    }
}
void scheduleRoundRobin(struct Process processes[], int n, int quantum) {
    calculateWaitingTime(processes, n, quantum);
    calculateTurnaroundTime(processes, n);
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].process_id, processes[i].burst_time,
            processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\n");
}
int main() {
    int n, i, quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the time quantum: ");
    scanf("%d", &quantum);
    struct Process processes[MAX_PROCESSES];
    for (i = 0; i < n; i++) {
```

```
        printf("Process %d\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].process_id = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
    }
    scheduleRoundRobin(processes, n, quantum);
    return 0;
}
```

Priority

```
#include <stdio.h>
#define MAX_PROCESSES 10
struct Process {
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};
void calculateWaitingTime(struct Process processes[], int n) {
    int i, j;
    processes[0].waiting_time = 0;
    for (i = 1; i < n; i++) {
        processes[i].waiting_time = 0;
        for (j = 0; j < i; j++) {
            processes[i].waiting_time += processes[j].burst_time;
        }
    }
}
void calculateTurnaroundTime(struct Process processes[], int n) {
    int i;
    for (i = 0; i < n; i++) {
```

```
        processes[i].turnaround_time = processes[i].burst_time + processes[i].waiting_time;
    }
}
void schedulePriority(struct Process processes[], int n) {
    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);
    printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].process_id, processes[i].burst_time,
               processes[i].priority, processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\n");
}
int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[MAX_PROCESSES];
    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        printf("Priority: ");
        scanf("%d", &processes[i].priority);
        processes[i].process_id = i + 1;
    }
    schedulePriority(processes, n);
    return 0;
}
```

HRRN

```
#include <stdio.h>
```

```
#define MAX_PROCESSES 10
struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};
void calculateWaitingTime(struct Process processes[], int n) {
    int i;
    processes[0].waiting_time = 0;
    for (i = 1; i < n; i++) {
        int waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time - processes[i].arrival_time;
        processes[i].waiting_time = (waiting_time > 0) ? waiting_time : 0;
    }
}
void calculateTurnaroundTime(struct Process processes[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
    }
}
void calculateResponseRatio(struct Process processes[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        double response_ratio = (double)processes[i].turnaround_time / processes[i].burst_time;
        printf("Process %d: Response Ratio = %.2f\n", processes[i].process_id, response_ratio);
    }
}
void scheduleHRRN(struct Process processes[], int n) {
    calculateResponseRatio(processes, n);
```

```
    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);
    printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].process_id, processes[i].arrival_time,
                processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\n");
}
int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[MAX_PROCESSES];
    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].process_id = i + 1;
    }
    scheduleHRRN(processes, n);
    return 0;
}
```