PROGRAM 3: PRIORITY SCHEDULING

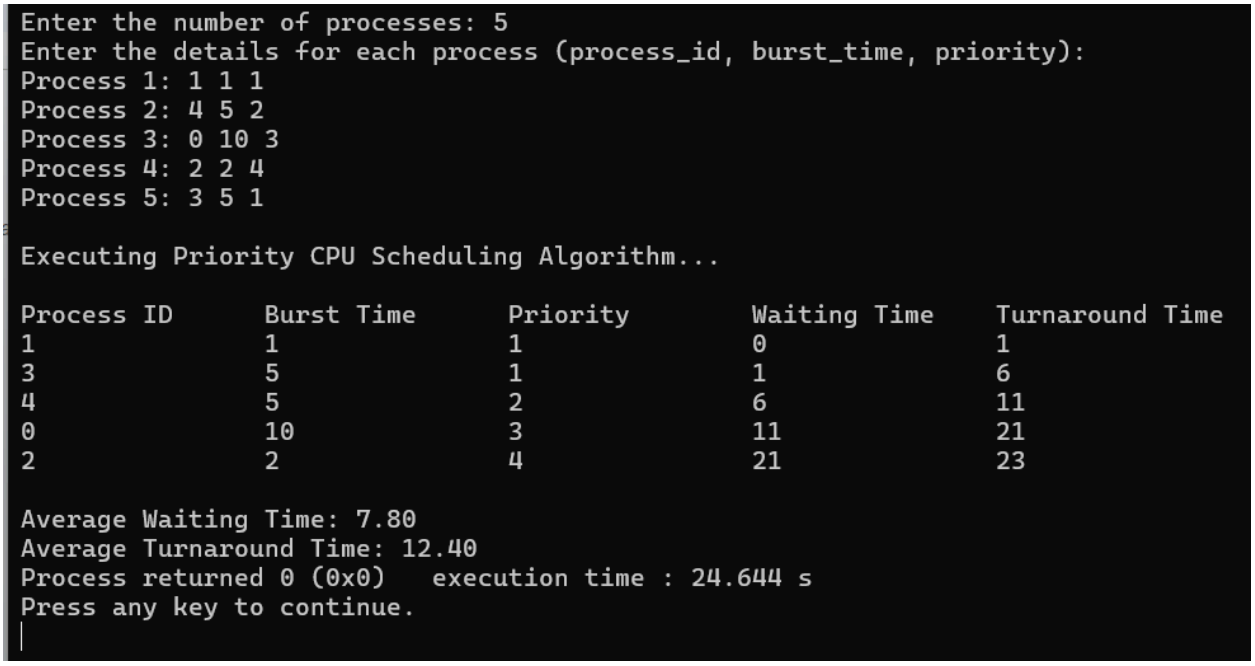
CODE:

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

struct Process {
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void calculate_times(struct Process *processes, int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;
    printf("\nProcess ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\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);
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
    printf("\nAverage Waiting Time: %.2f", (float)total_waiting_time / n);
    printf("\nAverage Turnaround Time: %.2f", (float)total_turnaround_time / n);
}

void priority_scheduling(struct Process *processes, int n) {
    // Sorting the processes by 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) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
```

```
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        // Calculate waiting time for the current process
        processes[i].waiting_time = current_time;
        // Update current time to include the burst time of the current process
        current_time += processes[i].burst_time;
        // Calculate turnaround time for the current process
        processes[i].turnaround_time = current_time;
    }

    calculate_times(processes, n);
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process *processes = (struct Process *)malloc(n * sizeof(struct Process));
    if (processes == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    printf("Enter the details for each process (process_id, burst_time, priority):\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d %d", &processes[i].process_id, &processes[i].burst_time,
&processes[i].priority);
        processes[i].waiting_time = 0;
        processes[i].turnaround_time = 0;
    }

    printf("\nExecuting Priority CPU Scheduling Algorithm...\n");
    priority_scheduling(processes, n);

    // Free dynamically allocated memory
    free(processes);
```

```
    return 0;
}
```

OUTPUT:



PROGRAM 4: ROUND ROBIN

CODE:

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

struct Process {
    int process_id;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_times(struct Process *processes, int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;
    printf("\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("%d\t\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);
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }
    printf("\nAverage Waiting Time: %.2f", (float)total_waiting_time / n);
    printf("\nAverage Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

void round_robin(struct Process *processes, int n, int quantum) {
    int *remaining_time = (int *)malloc(n * sizeof(int));
    int *completed = (int *)calloc(n, sizeof(int)); // To track completed processes
    int current_time = 0;
    int remaining_processes = n;

    for (int i = 0; i < n; i++) {
        remaining_time[i] = processes[i].burst_time;
    }

    while (remaining_processes > 0) {
        for (int i = 0; i < n; i++) {
            if (completed[i] == 0 && remaining_time[i] > 0) {
                if (remaining_time[i] <= quantum) {
                    current_time += remaining_time[i];
                    processes[i].turnaround_time = current_time;
                    remaining_time[i] = 0;
                    completed[i] = 1;
                    remaining_processes--;
                } else {
                    current_time += quantum;
                    remaining_time[i] -= quantum;
                }
            }
        }
    }

    // Calculate waiting time
    for (int i = 0; i < n; i++) {
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
```

```
    }

    calculate_times(processes, n);

    free(remaining_time);
    free(completed);
}

int main() {
    int n, quantum;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the time quantum: ");
    scanf("%d", &quantum);

    struct Process *processes = (struct Process *)malloc(n * sizeof(struct Process));
    if (processes == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    printf("Enter burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].process_id = i + 1;
    }

    printf("\nExecuting Round Robin CPU Scheduling Algorithm...\n");
    round_robin(processes, n, quantum);

    free(processes);

    return 0;
}
```

OUTPUT:

```
Enter the number of processes: 3
Enter the time quantum: 3
Enter burst time for each process:
Process 1: 24
Process 2: 3
Process 3: 3

Executing Round Robin CPU Scheduling Algorithm...

Process ID      Burst Time      Waiting Time    Turnaround Time
1               24              6               30
2               3               3               6
3               3               6               9

Average Waiting Time: 5.00
Average Turnaround Time: 15.00

Process returned 0 (0x0)   execution time : 6.450 s
Press any key to continue.
```