LAB PROGRAM 4

Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) Rate- Monotonic

b) Earliest-deadline First

c) Proportional scheduling

INPUT

```c
#include<stdio.h>
typedef struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int deadline;
    float priority;
} Process;

void rate_monotonic(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].deadline > processes[j + 1].deadline) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    printf("Rate-Monotonic scheduling:\n");
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        printf("Process %d executes from time %d to %d\n", processes[i].id, current_time, current_time + processes[i].burst_time);
        current_time += processes[i].burst_time;
    }
}

void earliest_deadline_first(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].deadline > processes[j + 1].deadline) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    printf("\nEarliest-Deadline First scheduling:\n");
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        printf("Process %d executes from time %d to %d\n", processes[i].id, current_time, current_time + processes[i].burst_time);
        current_time += processes[i].burst_time;
    }
}

void proportional_scheduling(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
```

```
    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) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    printf("\nProportional scheduling:\n");
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        printf("Process %d executes from time %d to %d\n", processes[i].id, current_time, current_time + processes[i].burst_time);
        current_time += processes[i].burst_time;
    }
}

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

    Process processes[n];
    for (int i = 0; i < n; i++) {
        printf("Enter details for Process %d:\n", i + 1);
        processes[i].id = i + 1;
        printf("Arrival time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst time: ");
        scanf("%d", &processes[i].burst_time);
        printf("Deadline: ");
        scanf("%d", &processes[i].deadline);
        printf("Priority: ");
        scanf("%f", &processes[i].priority);
    }

    rate_monotonic(processes, n);
    earliest_deadline_first(processes, n);
    proportional_scheduling(processes, n);

    return 0;
}
```

OUTPUT

```
Deadline: 2
Priority: 1
Enter details for Process 2:
Arrival time: 1
Burst time: 5
Deadline: 2
Priority: 1
Enter details for Process 3:
Arrival time: 4
Burst time: 2
Deadline: 5
Priority: 3
Rate-Monotonic scheduling:
Process 1 executes from time 0 to 3
Process 2 executes from time 3 to 8
Process 3 executes from time 8 to 10

Earliest-Deadline First scheduling:
Process 1 executes from time 0 to 3
Process 2 executes from time 3 to 8
Process 3 executes from time 8 to 10

Proportional scheduling:
Process 3 executes from time 0 to 2
Process 1 executes from time 2 to 5
Process 2 executes from time 5 to 10

Process returned 0 (0x0)   execution time : 31.694 s
Press any key to continue.
```