GitHub link :
https://github.com/Lavkumarsharma/CSE_316_12111673

NAME – LAW KUMAR
REG: 12111673
ROLL NO: K21XRB20
SECTION: K21XR

Q2. Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as :
Priority = 1+ Waiting time / Estimated run time
Write a program to implement such an algorithm. Ensure
1. The input is given dynamically at run time by the user
2. The priority of each process is visible after each unit of time
3. The gantt chart is shown as an output
4. Calculate individual waiting time and average waiting time

Solution:

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

```
#define MAX_JOBS 100

typedef struct {
    char name[20];
    int arrival_time;
    int estimated_run_time;
    int waiting_time;
} Job;

int compare_jobs(const void *a, const void *b) {
    Job *job_a = (Job *)a;
    Job *job_b = (Job *)b;
    float priority_a = 1.0 + (float)job_a->waiting_time / job_a->estimated_run_time;
    float priority_b = 1.0 + (float)job_b->waiting_time / job_b->estimated_run_time;
    if (priority_a < priority_b) {
        return -1;
    } else if (priority_a > priority_b) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    int n;
    Job jobs[MAX_JOBS];
    int time = 0;
    int num_completed_jobs = 0;
    int waiting_times[MAX_JOBS] = {0};
    char gantt_chart[MAX_JOBS * 2 + 1] = {'\0'};

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

    for (int i = 0; i < n; i++) {
        printf("Enter the name of job %d: ", i + 1);
        scanf("%s", jobs[i].name);
        printf("Enter the arrival time of job %d: ", i + 1);
        scanf("%d", &jobs[i].arrival_time);
        printf("Enter the estimated run time of job %d: ", i + 1);
        scanf("%d", &jobs[i].estimated_run_time);
        jobs[i].waiting_time = 0;
    }

    while (num_completed_jobs < n) {
        qsort(jobs, n, sizeof(Job), compare_jobs);

        Job *current_job = &jobs[0];
        current_job->waiting_time += time - current_job->arrival_time;
        waiting_times[num_completed_jobs] = current_job->waiting_time;

        strcat(gantt_chart, current_job->name);
        strcat(gantt_chart, " ");

        for (int i = 0; i < current_job->estimated_run_time; i++) {
            time++;
        }

        num_completed_jobs++;
    }

    float total_waiting_time = 0.0;
    printf("Individual Waiting Times: ");
    for (int i = 0; i < n; i++) {
        total_waiting_time += waiting_times[i];
        printf("%d ", waiting_times[i]);
    }
    printf("\nAverage Waiting Time: %.2f\n", total_waiting_time / n);
    printf("Gantt Chart: %s\n", gantt_chart);

    return 0;
}
```

OUTPUT:

```
Enter the number of jobs: 4
Enter the name of job 1: job_1
Enter the arrival time of job 1: 0
Enter the estimated run time of job 1: 7
Enter the name of job 2: job_2
Enter the arrival time of job 2: 2
Enter the estimated run time of job 2: 5
Enter the name of job 3: job_3
Enter the arrival time of job 3: 0
Enter the estimated run time of job 3: 9
Enter the name of job 4: job_4
Enter the arrival time of job 4: 8
Enter the estimated run time of job 4: 3
Individual Waiting Times: 0 7 12 19
Average Waiting Time: 9.50
Gantt Chart: job_1 job_1 job_2 job_3
```