EDF

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

#define arrival 0
#define execution 1
#define deadline 2
#define period 3
#define abs_arrival 4
#define execution_copy 5
#define abs_deadline 6

typedef struct
{
    int T[7], instance, alive;

} task;

#define IDLE_TASK_ID 1023
#define ALL 1
#define CURRENT 0

void get_tasks(task *t1, int n);
int hyperperiod_calc(task *t1, int n);
float cpu_util(task *t1, int n);
int gcd(int a, int b);
int lcm(int *a, int n);
int sp_interrupt(task *t1, int tmr, int n);
int min(task *t1, int n, int p);
void update_abs_arrival(task *t1, int n, int k, int all);
void update_abs_deadline(task *t1, int n, int all);
```

```
void copy_execution_time(task *t1, int n, int all);

int timer = 0;

int main()
{
    task *t;
    int n, hyper_period, active_task_id;
    float cpu_utilization;
    printf("Enter number of tasks\n");
    scanf("%d", &n);
    t = (task *)malloc(n * sizeof(task));
    get_tasks(t, n);
    cpu_utilization = cpu_util(t, n);
    printf("CPU Utilization %f\n", cpu_utilization);

    if (cpu_utilization < 1)
        printf("Tasks can be scheduled\n");
    else
        printf("Schedule is not feasible\n");

    hyper_period = hyperperiod_calc(t, n);
    copy_execution_time(t, n, ALL);
    update_abs_arrival(t, n, 0, ALL);
    update_abs_deadline(t, n, ALL);

    while (timer < hyper_period)
    {
        ++timer;
        if (timer < 10)
            printf("|  %d", timer);
        else
            printf("| %d", timer);
    }
    printf("|\n");
```

```
    timer = 0;
    while (timer < hyper_period)
    {

        if (sp_interrupt(t, timer, n))
        {
            active_task_id = min(t, n, abs_deadline);
        }

        if (active_task_id == IDLE_TASK_ID)
        {
            printf("|---");
        }

        if (active_task_id != IDLE_TASK_ID)
        {

            if (t[active_task_id].T[execution_copy] != 0)
            {
                t[active_task_id].T[execution_copy]--;
                printf("|T-%d", active_task_id + 1);
            }

            if (t[active_task_id].T[execution_copy] == 0)
            {
                t[active_task_id].instance++;
                t[active_task_id].alive = 0;
                copy_execution_time(t, active_task_id, CURRENT);
                update_abs_arrival(t, active_task_id, t[active_task_id].instance,
CURRENT);
                update_abs_deadline(t, active_task_id, CURRENT);
                active_task_id = min(t, n, abs_deadline);
            }
        }
```

```
        ++timer;
    }
    printf("|\n");
    free(t);
    return 0;
}

void get_tasks(task *t1, int n)
{
    int i = 0;
    while (i < n)
    {
        printf("Enter Task %d parameters\n", i + 1);
        t1->T[arrival] = 0;
        printf("Execution time: ");
        scanf("%d", &t1->T[execution]);
        printf("Deadline time: ");
        scanf("%d", &t1->T[deadline]);
        printf("Period: ");
        scanf("%d", &t1->T[period]);
        t1->T[abs_arrival] = 0;
        t1->T[execution_copy] = 0;
        t1->T[abs_deadline] = 0;
        t1->instance = 0;
        t1->alive = 0;
        t1++;
        i++;
    }
}

int hyperperiod_calc(task *t1, int n)
{
    int i = 0, ht, a[10];
    while (i < n)
```

```
    {
        a[i] = t1->T[period];
        t1++;
        i++;
    }
    ht = lcm(a, n);

    return ht;
}

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}

int lcm(int *a, int n)
{
    int res = 1, i;
    for (i = 0; i < n; i++)
    {
        res = res * a[i] / gcd(res, a[i]);
    }
    return res;
}

int sp_interrupt(task *t1, int tmr, int n)
{
    int i = 0, n1 = 0, a = 0;
    task *t1_copy;
    t1_copy = t1;
    while (i < n)
    {
```

```
        if (tmr == t1->T[abs_arrival])
        {
            t1->alive = 1;
            a++;
        }
        t1++;
        i++;
    }

    t1 = t1_copy;
    i = 0;

    while (i < n)
    {
        if (t1->alive == 0)
            n1++;
        t1++;
        i++;
    }

    if (n1 == n || a != 0)
    {
        return 1;
    }

    return 0;
}

void update_abs_deadline(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
```

```
            t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
    }
}

void update_abs_arrival(task *t1, int n, int k, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
    }
}

void copy_execution_time(task *t1, int n, int all)
{
    int i = 0;
    if (all)
```

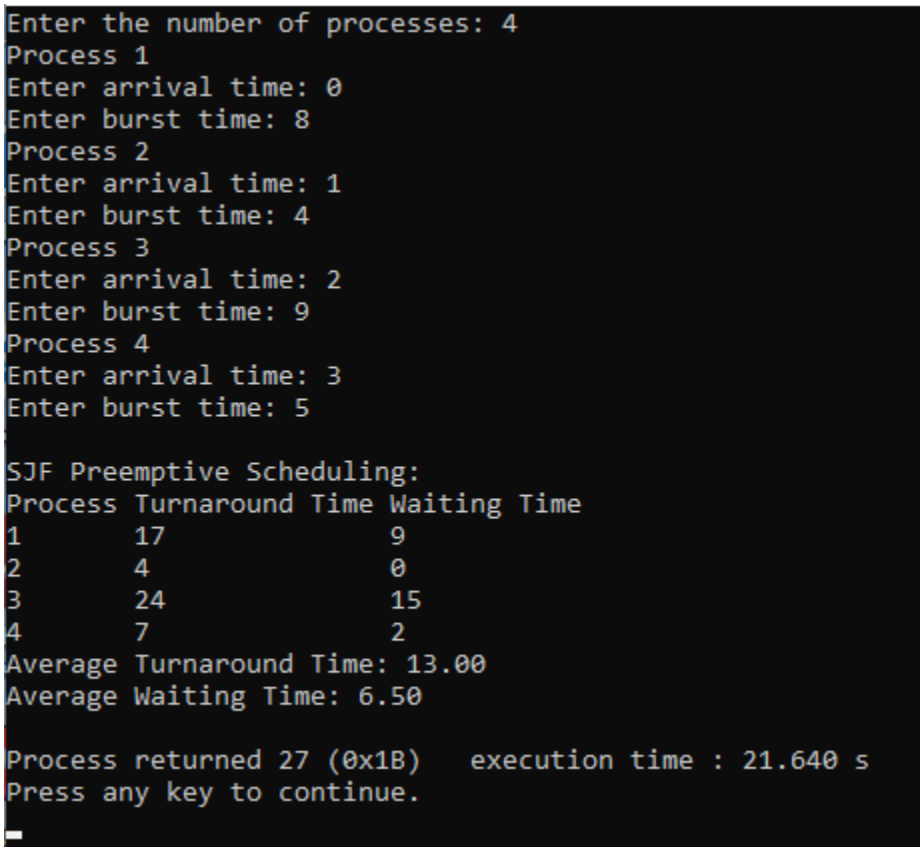
```
    {
        while (i < n)
        {
            t1->T[execution_copy] = t1->T[execution];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[execution_copy] = t1->T[execution];
    }
}

int min(task *t1, int n, int p)
{
    int i = 0, min = 0x7FFF, task_id = IDLE_TASK_ID;
    while (i < n)
    {
        if (min > t1->T[p] && t1->alive == 1)
        {
            min = t1->T[p];
            task_id = i;
        }
        t1++;
        i++;
    }
    return task_id;
}

float cpu_util(task *t1, int n)
{
    int i = 0;
    float cu = 0;
```

```
    while (i < n)
    {
        cu = cu + (float)t1->T[execution] / (float)t1->T[deadline];
        t1++;
        i++;
    }
    return cu;
}
```

OUTPUT:



```
Enter the number of processes: 4
Process 1
Enter arrival time: 0
Enter burst time: 8
Process 2
Enter arrival time: 1
Enter burst time: 4
Process 3
Enter arrival time: 2
Enter burst time: 9
Process 4
Enter arrival time: 3
Enter burst time: 5

SJF Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       17              9
2       4               0
3       24              15
4       7               2
Average Turnaround Time: 13.00
Average Waiting Time: 6.50

Process returned 27 (0x1B)   execution time : 21.640 s
Press any key to continue.
```