Simulate Rate Monotonic Scheduling for the following and show the order of execution of processes in CPU timeline

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

#define MAX_PROCESS 10

int num_of_process = 3;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
remain_time[MAX_PROCESS];

// collecting details of processes
void get_process_info()
{
   printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
   scanf("%d", &num_of_process);
   if (num_of_process < 1)
   {
      printf("Do you really want to schedule %d processes? -_-\n", num_of_process);
      exit(0);
   }
   for (int i = 0; i < num_of_process; i++)
   {
      printf("\nProcess %d:-\n", i + 1);

      printf("==> Execution time: ");
      scanf("%d", &execution_time[i]);
      remain_time[i] = execution_time[i];
```

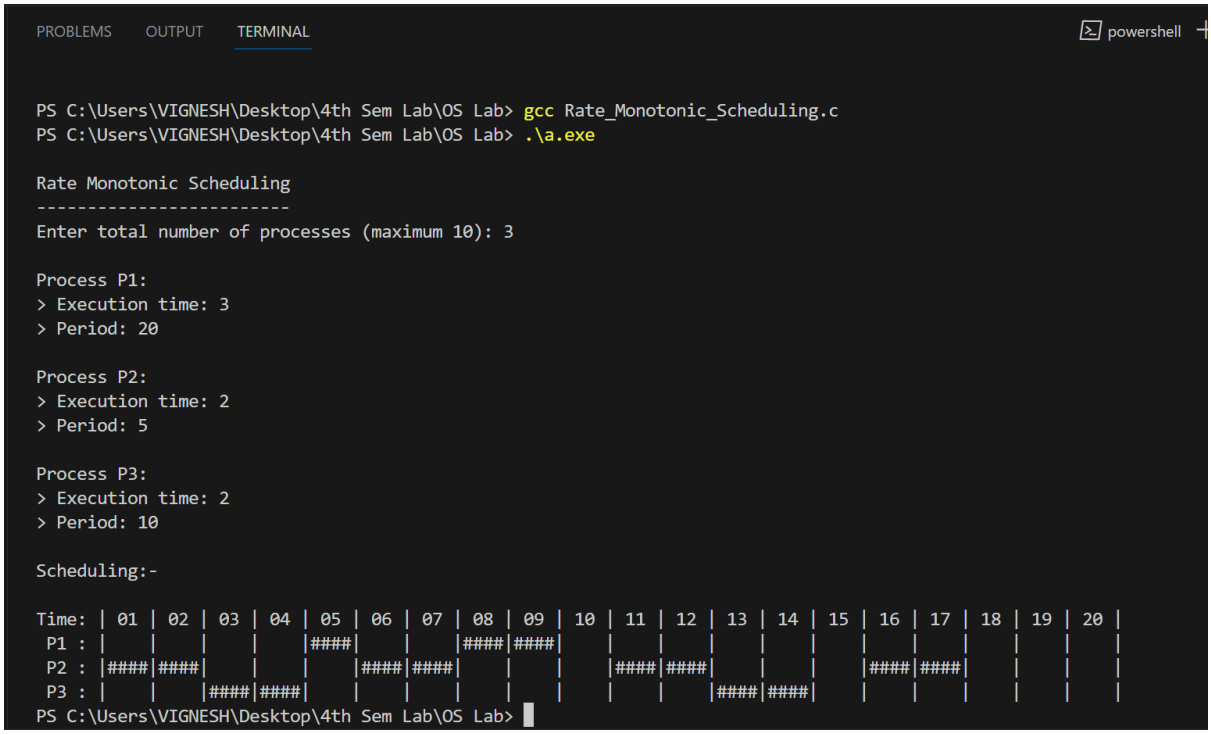
```
        printf("==> Period: ");
        scanf("%d", &period[i]);
    }
}

// get maximum of three numbers
int max(int a, int b, int c)
{
    if (a >= b && a >= c)
        return a;
    else if (b >= a && b >= c)
        return b;
    else
        return c;
}

// calculating the observation time for scheduling timeline
int get_observation_time()
{
    return max(period[0], period[1], period[2]);
}

// print scheduling sequence
void print_schedule(int process_list[], int cycles)
{
    printf("\nScheduling:-\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++)
    {
        if (i < 9)
```

```
            printf("| 0%d ", i + 1);
        else
            printf("| %d ", i + 1);
    }
    printf("|\n");

    for (int i = 0; i < num_of_process; i++)
    {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++)
        {
            if (process_list[j] == i + 1)
                printf("|####");
            else
                printf("|    ");
        }
        printf("|\n");
    }
}

void rate_monotonic(int time)
{
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++)
    {
        utilization += (1.0 * execution_time[i]) / period[i];
    }
    int n = num_of_process;
    if (utilization > n * (pow(2, 1.0 / n) - 1))
    {
```

```
    printf("\nGiven problem is not schedulable under said scheduling algorithm.\n");
    exit(0);
}

int process_list[time];
int min = 999, next_process = 0;
for (int i = 0; i < time; i++)
{
    min = 1000;
    for (int j = 0; j < num_of_process; j++)
    {
        if (remain_time[j] > 0)
        {
            if (min > period[j])
            {
                min = period[j];
                next_process = j;
            }
        }
    }
    if (remain_time[next_process] > 0)
    {
        process_list[i] = next_process + 1; // +1 for catering 0 array index.
        remain_time[next_process] -= 1;
    }

    for (int k = 0; k < num_of_process; k++)
    {
        if ((i + 1) % period[k] == 0)
        {
```

```
                remain_time[k] = execution_time[k];
                next_process = k;
            }
        }
    }
    print_schedule(process_list, time);
}

int main(int argc, char *argv[])
{
    printf("Rate Monotonic Scheduling\n");
    printf("-------------------------\n");

    get_process_info(); // collecting processes detail
    int observation_time = get_observation_time();

    rate_monotonic(observation_time);

    return 0;
}
```

OUTPUT:


PROBLEMS
OUTPUT
TERMINAL
powershell
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> gcc Rate_Monotonic_Scheduling.c
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> .\a.exe
Rate Monotonic Scheduling
Enter total number of processes (maximum 10): 3
Process P1:
> Execution time: 3
> Period: 20
Process P2:
> Execution time: 2
> Period: 5
Process P3:
> Execution time: 2
> Period: 10
Scheduling:-
Time: | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
P1 : | | | | | #### | | | #### | #### | | | | | | | | | | | |
P2 : | #### | #### | | | | #### | #### | | | | #### | #### | | | | #### | #### | | | |
P3 : | | | #### | #### | | | | | | | | | #### | #### | | | | | | |
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab>

Simulate Earliest Deadline First for the following and show the order of execution of processes in CPU timeline:

```
#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("|Idl");
    }

    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:

```
PROBLEMS    OUTPUT    TERMINAL

PS C:\Users\VIGNESH\Desktop\OSLAB> gcc EDF.c
PS C:\Users\VIGNESH\Desktop\OSLAB> .\a.exe
Enter number of tasks
3
Enter Task 1 parameters
Execution time: 3
Deadline time: 7
Period: 20
Enter Task 2 parameters
Execution time: 2
Deadline time: 4
Period: 5
Enter Task 3 parameters
Execution time: 2
Deadline time: 8
Period: 10
CPU Utilization 1.178571
Schedule is not feasible
|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
|T-2|T-2|T-1|T-1|T-1|T-3|T-3|T-2|T-2|---|T-2|T-2|T-3|T-3|---|T-2|T-2|---|---|---|
PS C:\Users\VIGNESH\Desktop\OSLAB>
```