Program-3
Write a C program to simulate Real-Time CPU Scheduling algorithms a) Rate- Monotonic

Code:

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

#define MAX_PROCESS 10

typedef struct {
   int
id;
int
burst_
time;
float
priorit
y;
} Task;
int num_of_process; int execution_time[MAX_PROCESS],
period[MAX_PROCESS], remain_time[MAX_PROCESS],
deadline[MAX_PROCESS], remain_deadline[MAX_PROCESS];

void get_process_info(int selected_algo)
{
   printf("Enter total number of processes (maximum %d): ",
MAX_PROCESS);     scanf("%d", &num_of_process);     if
(num_of_process < 1)
   {
      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];        if
(selected_algo == 2)
      {
         printf("==> Deadline:
");           scanf("%d",
&deadline[i]);
      }
      else
      {
         printf("==> Period:
");           scanf("%d",
&period[i]);        }
   }
}

int max(int a, int b, int c) {
   int max;
if (a >= b && a
>= c)        max
= a;
else if (b >= a
&& b >= c)
max = b;
else if (c >= a
&& c >= b)

max =
c;
return
max;
}

int get_observation_time(int selected_algo)
{
   if (selected_algo == 1)
   {
      return max(period[0], period[1], period[2]);
```

```
    }
    else if (selected_algo == 2)
    {
        return max(deadline[0], deadline[1], deadline[2]);
    }
}

int uti_time=0; void
ut_time(int
selected_algo){
if(selected_algo==1)
  {
      for (int i = 0; i < num_of_process; i++)
    {
      uti_time+=(execution_time[i]/period[i]);
  }
}
    else if(selected_algo==2)
  {
      for (int i = 0; i < num_of_process; i++)
    {
      uti_time+=(execution_time[i]/deadline[i]);
  }
}
}

void print_schedule(int process_list[], int cycles)
{
    printf("\nScheduling:\n\n");
    printf("Time:
");
for (int i = 0; i <
cycles; i++){
if (i < 10)

printf("| 0%d
", i);         else
```

```
printf("| %d ",
i);
    }
    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)
{
    int process_list[100] = {0}, min = 999,
next_process = 0;     float utilization = 0.0;
for (int i = 0; i < num_of_process; i++)
    {
        utilization += (float)(execution_time[i] / period[i]);
    }
    int n =
num_of_process;
float m = (n * (pow(2, 1.0
/ n) - 1));     if (utilization
> m)
    {
        printf("\nGiven problem is not schedulable under the said scheduling algorithm.\n");
```

```
    }
    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;
            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;
            }
        }
    }
    printf("Utilisation time %d",utilization);
print_schedule(process_list, time);
}

void earliest_deadline_first(int time){
float utilization = 0;
 for (int i = 0; i < num_of_process; i++){
```

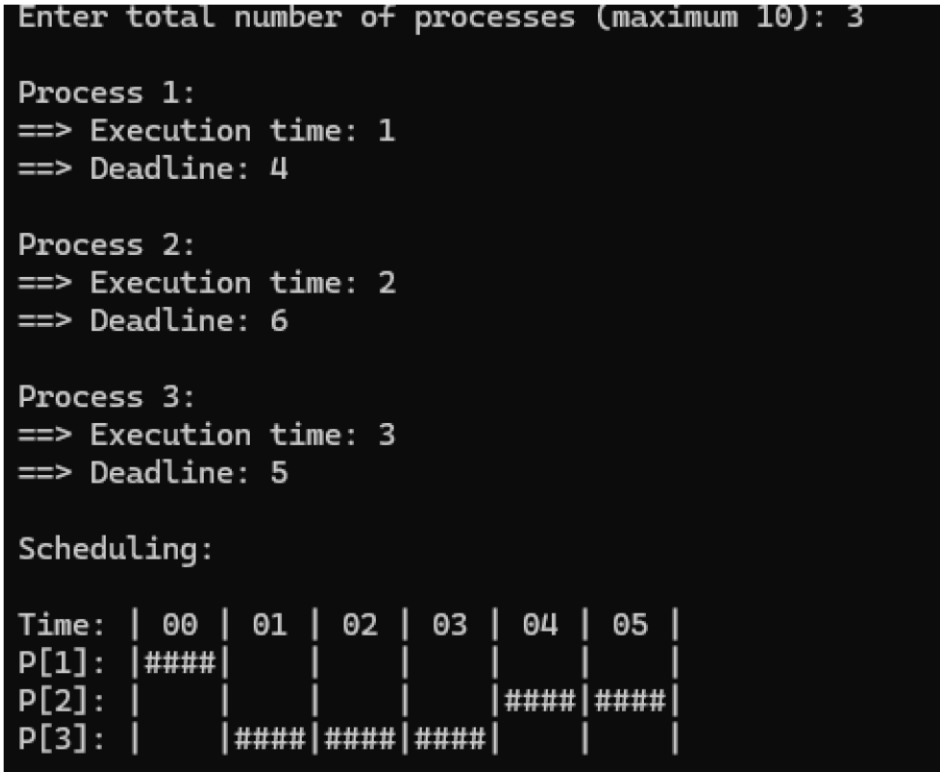
```
utilization +=
(1.0*execution_time[i])/deadline[i];
    }
    int n = num_of_process;
    int process[num_of_process];
int max_deadline, current_process=0,
min_deadline,process_list[time];     bool
is_ready[num_of_process];


    for(int i=0; i<num_of_process; i++){
        is_ready[i] =
true;         process[i]
= i+1;
    }


max_deadline=deadline[0]
;     for(int i=1;
i<num_of_process; i++){
if(deadline[i] >
max_deadline)
max_deadline =
deadline[i];
    }
    for(int i=0; i<num_of_process;
i++){         for(int j=i+1;
j<num_of_process; j++){
if(deadline[j] < deadline[i]){
 int temp = execution_time[j];
execution_time[j] =
execution_time[i];
execution_time[i] = temp;
 temp = deadline[j];
 deadline[j] = deadline[i];
  deadline[i] = temp;
temp = process[j];
  process[j] = process[i];
   process[i] = temp;
           }
        }
```

```
    }
    for(int i=0;
i<num_of_process; i++){
remain_time[i] =
execution_time[i];
remain_deadline[i] =
deadline[i];
    }
    for (int t = 0; t < time; t++){
if(current_process != -1){
            --execution_time[current_process];
            process_list[t] = process[current_process];
        }
else


process_list[t]
=0;
        for(int i=0;i<num_of_process;i++){
            --deadline[i];
 if((execution_time[i] == 0) &&
is_ready[i]){                deadline[i] +=
remain_deadline[i];
is_ready[i] = false;
            }
            if((deadline[i] <= remain_deadline[i]) && (is_ready[i] == false)){
      execution_time[i] = remain_time[i];
                is_ready[i] = true;
            }
        }
        min_deadline = max_deadline;
  current_process = -1;
  for(int i=0;i<num_of_process;i++){
     if((deadline[i] <= min_deadline) &&
(execution_time[i] > 0)){                current_process = i;
                min_deadline = deadline[i];
            }
        }
    }
```

```
    print_schedule(process_list, time);
}

int main()
{
    int option;
    int observation_time;
    while (1)
    {
    printf("\n1. Rate Monotonic\n2. Earliest Deadline first\n3. Proportional
Scheduling\n\nEnter your choice: ");
  scanf("%d", &option);
  switch(option){
        case 1: get_process_info(option);
observation_time =
get_observation_time(option);
rate_monotonic(observation_time);
 break;
        case 2: get_process_info(option);
                observation_time =
get_observation_time(option);
earliest_deadline_first(observation_time);
        break;


        case 3: exit (0);
  default: printf("\nInvalid
Statement");
    }
    }
    return 0;
}
```

Output:

```
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 1
==> Period: 4

Process 2:
==> Execution time: 2
==> Period: 6
```


Enter total number of processes (maximum 10): 3
Process 1:
==> Execution time: 1
==> Deadline: 4
Process 2:
==> Execution time: 2
==> Deadline: 6
Process 3:
==> Execution time: 3
==> Deadline: 5
Scheduling:
Time: | 00 | 01 | 02 | 03 | 04 | 05 |
P[1]: |####| | | | | |
P[2]: | | | | |####|####|
P[3]: | |####|####|####| | |