Write a C program to simulate Real-Time CPU Scheduling algorithms:
(a) Rate- Monotonic
(b) Earliest-deadline First
(c) Proportional scheduling

1.1.1 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 priority;

} 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]);
        }
}

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;
        for (int i = 0; i < num_of_process; i++)
```

```
{
utilization += (1.0 * execution_time[i]) / period[i];
}
int n = num_of_process;
int m = (float) (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;
        }
        }
        }
        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);
}

void proportionalScheduling() {
        int n;
        printf("Enter the number of tasks: ");
        scanf("%d", &n);

        Task tasks[n];
        printf("Enter burst time and priority for each task:\n");
        for (int i = 0; i < n; i++) {
        tasks[i].id = i + 1;
        printf("Task %d - Burst Time: ", tasks[i].id);
        scanf("%d", &tasks[i].burst_time);
```

```
printf("Task %d - Priority: ", tasks[i].id);
scanf("%f", &tasks[i].priority);
}

// Sort tasks based on priority (ascending order)
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (tasks[j].priority > tasks[j + 1].priority) {
        // Swap tasks
        Task temp = tasks[j];
        tasks[j] = tasks[j + 1];
        tasks[j + 1] = temp;
}
}
}

printf("\nProportional Scheduling:\n");

int total_burst_time = 0;
float total_priority = 0.0;
```

```
    for (int i = 0; i < n; i++) {
    total_burst_time += tasks[i].burst_time;
    total_priority += tasks[i].priority;
    }

    for (int i = 0; i < n; i++) {
    float time_slice = (tasks[i].priority / total_priority) * total_burst_time;
    printf("Task %d executes for %.2f units of time\n", tasks[i].id, time_slice);
    }
}

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: proportionalScheduling();
                break;
        case 4: exit (0);
        default: printf("\nInvalid Statement");
        }
        }
        return 0;
}
```

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 1
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

Process 2:
==> Execution time: 2
==> Period: 5

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
P[2]: |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
P[3]: |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |
```

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 2
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Deadline: 7

Process 2:
==> Execution time: 2
==> Deadline: 4

Process 3:
==> Execution time: 2
==> Deadline: 8

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
P[1]: |    |    |####|####|####|    |    |    |
P[2]: |####|####|    |    |    |    |    |####|
P[3]: |    |    |    |    |    |####|####|    |
```

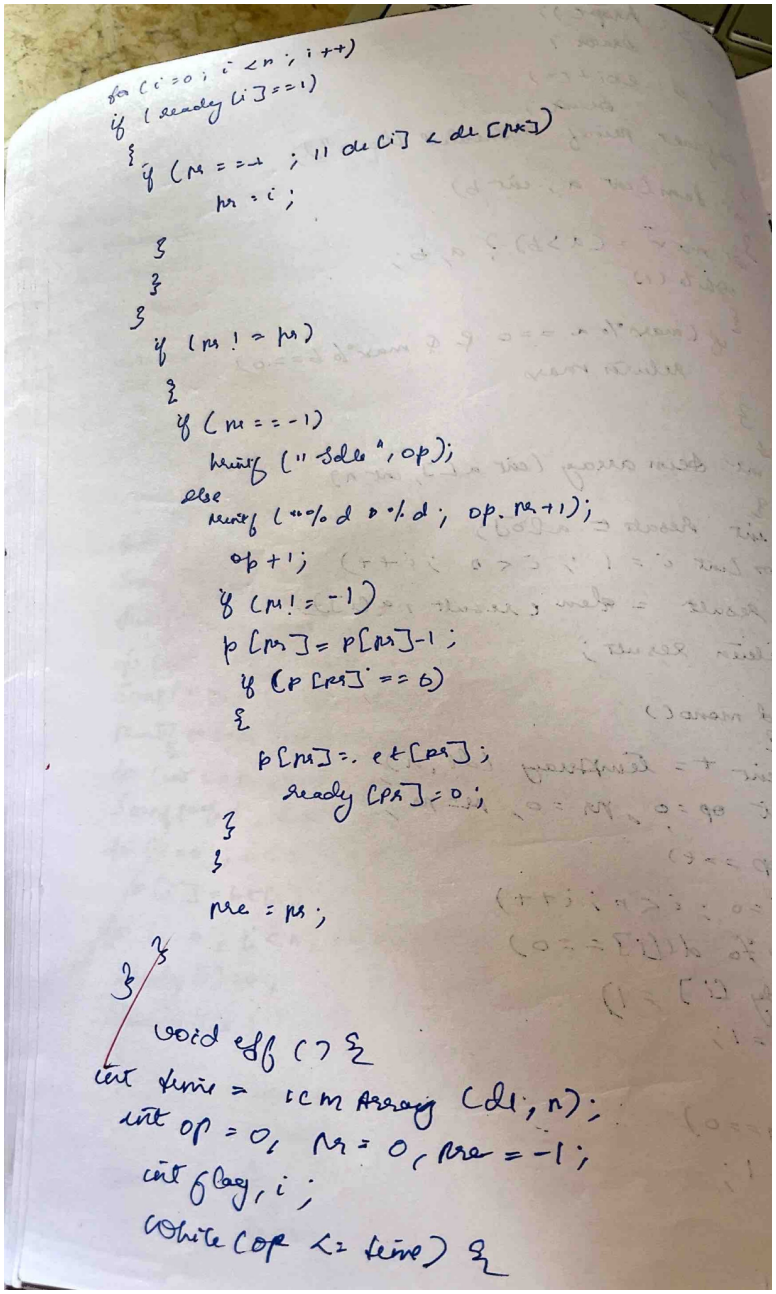
```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 3
Enter the number of tasks: 3
Enter burst time and priority for each task:
Task 1 - Burst Time: 4
Task 1 - Priority: 2
Task 2 - Burst Time: 6
Task 2 - Priority: 3
Task 3 - Burst Time: 5
Task 3 - Priority: 1

Proportional Scheduling:
Task 3 executes for 2.50 units of time
Task 1 executes for 5.00 units of time
Task 2 executes for 7.50 units of time
```

1) Rate monotonic
2) Earliest deadline first
3) Proportional Scheduling

```
#include <stdio.h>
#include <stdlib.h>
int [20], n, i, d[10], p[10], ready[10], flag1;
int main()
{
  int ~~main()~~ i, k=1;
  while(k)
  {
  printf("1. Monotonic \n 2. Earliest deadline \n 3.
        Proportional \n 4. Exit");
  Scanf("%d", &ch);
  printf("Enter no of process");
  Scanf("%d", &n);
  printf("Enter execution time");
  for(int i=0; i<n; i++);
  Scanf("%d", &bt[i])
  printf("Enter deadline");
  for(int i=0; i<n; i++)
  Scanf("%d", &di[i]);
  for(i=0; i<n; i++)
    p[i] = bt[i];
  for(i=0; i<n; i++)
    ready[i]=0;
  Switch(ch)
  {
  Case 1: mon()
          break;

Case 2: edf()
          break;
```

```
case 3: hrp();
        break;
case 4: exit(0)
        break;
default: printf("Invalid
}
int dem(int a, int b)
{
int max = (a>b)?
while(1)
{
  if(max%a ==
    return max
}
int dem arra
{
  int Result
  for(int i =
    result =
  return re
}
void mon
{
  int
  int
while(op
{ for(i
  if(
  re
}
if
```

```
case 3 : hcf();
         break;
case 4 : exit(0)
         break;
default printf("Invalid choice");
}
int lcm(int a, int b)
{
int max = (a>b) ? a, b;
while (1)
{
if (max % a == 0 && max % b == 0)
    return max
}
}
int lcm array (int a[], int n)
{
int result = a[0];
for (int i = 1; i < n ; i++)
  result = lcm (result, a[i]);
return result;
}
void main()
  {
  int * = temparray (d1, n);
  int op = 0, pr = 0, re = pr;
while (op == t)
{ for (i=0 ; i<n; i++)
  if (op % d1[i] == 0)
    ready [i] = 1)
    flag = 1;
}
  if (flag==0)
    pr = -1;
  else
  {
    pr = -1
```



```
for (i=0; i<n; i++)
if (ready[i]==1)
{
 if (ns==-1 ; || dt[i] < dt[ns])
      ns = i;

  }
  }
 }
  if (ns != ps)
   {
    if (ps==-1)
      printf ("Idle", op);
    else
      printf ("%d %d", op, ps+1);
       op++;
       if (ps!=-1)
       p[ps] = p[ps]-1;
        if (p[ps] == 0)
        {
          p[ps] = et[ps];
           ready[ps] = 0;
       }
       }

       pre = ps;

}
}

void eff () {
int time = lcm Array (dt, n);
int op = 0, ns = 0, pre = -1;
int flag, i;
while (op <= time) {
```

```
for(i=0; i<n; i++){
 if (cop % d1[i] == 0) {
    ready[i] = 1;
  }

 }
 flag = 0;
 for(i=0; i<n; i++) {
 if (ready[i] == 1) {
  flag = 1;
 break;
 }
}
 if (flag == 0) {
    ns = -1;
 } else {
 ns = -1;
 for (i=0; i<n; i++) {
  if (ready[i] == 1) {
   if (ns == -1 || p[i] < p[ns]) {
      ns = i;
   }
  }
 }
}
 if (pre != ns) {
  if (ns == -1) {
   printf ("%d Idle", cop);
  } else {
   printf ("%d %d", cop, ns+1);
 }
}
 cop++;
```

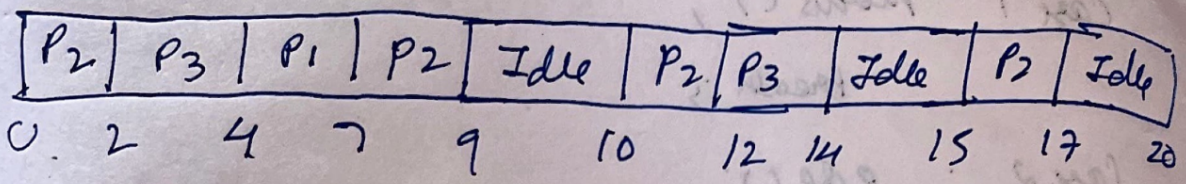
```
if (pa != -1) {
    p[pa] = p[pa] - 1;
    if (p[pa] == 0) {
    t[pa] = et[pa];
    ready[pa] = 0;
  }
}
  me = pa;
}
  printf("\n");
}

void prop() {
}
int main() {
int ch, k = 1;
while(k) {
printf("Enter your choice: n1. Monotonic \n2. Edf
n3. Proportional \n4. Exit\n");
scanf("%d", &ch);
if(ch == 4)
  exit(0);
printf("Enter no of processes");
Scanf("%d", &n);
printf("Enter execution times:\n");
for(i = 0; i < n; i++)
  scanf("%d", &et[i]);
printf("Enter deadlines:\n");
for(i = 0; i < n; i++)
  scanf("%d", &dl[i]);
  for(i = 0; i < n; i++)
    p[i] = et[i];
```

Enter no of process : 3

enter execution times: 3 2 2

enter deadlines: 20 5 10

0 P_2 2 P_3 4 P_1 7 P_2 9 Idle 10 P_2 12 P_3 14
Idle 15 P_2 17 Idle 20 P_2



9/10

26/7/23