4.Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

Code:

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


int et[10], i, n, dl[10], p[10], ready[10], flag = 1;

int lcm(int a, int b) {
    int max = (a > b) ? a : b;
    while (1) {
        if (max % a == 0 && max % b == 0)
            return max;
        max++;
    }
}

int lcmArray(int arr[], int n) {
    int result = arr[0];
    for (int i = 1; i < n; i++) {
        result = lcm(result, arr[i]);
    }
    return result;
}

void mono() {
    int time = lcmArray(dl, n);
    int op = 0, pr = 0, pre = pr;



    while (op <= time) {
        for (i = 0; i < n; i++) {
            if (op % dl[i] == 0) {
                ready[i] = 1;
            }
        }

        flag = 0;
```

```
        for (i = 0; i < n; i++) {
            if (ready[i] == 1) {
                flag = 1;
                break;
            }
        }

        if (flag == 0) {
            pr = -1;
        } else {
            pr = -1;
            for (i = 0; i < n; i++) {
                if (ready[i] == 1) {
                    if (pr == -1 || dl[i] < dl[pr]) {
                        pr = i;
                    }
                }
            }
        }


            if (pr != pre) {
                if (pr == -1) {
                    printf("%d Idle ",op);
                } else {
                    printf("%d P%d ",op, pr + 1);
                }
            }

        op++;
        if (pr != -1) {
            p[pr] = p[pr] - 1;
            if (p[pr] == 0) {
                p[pr] = et[pr];
                ready[pr] = 0;
            }
        }

        pre = pr;
    }

    printf("\n");
}
```

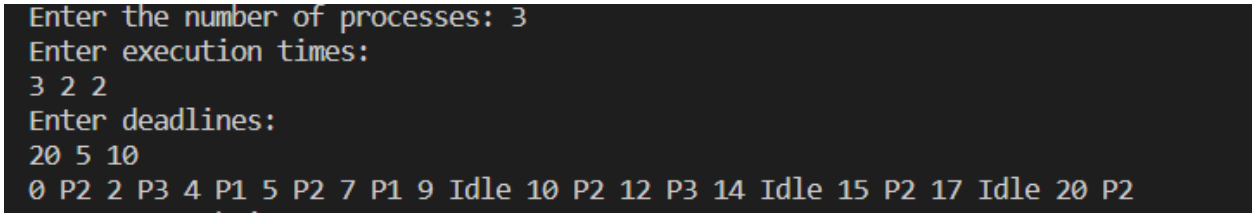
```
void edf() {
    int time = lcmArray(dl, n);
    int op = 0, pr = 0, pre = -1;
    int flag, i;

    while (op <= time) {
        for (i = 0; i < n; i++) {
            if (op % dl[i] == 0) {
                ready[i] = 1;
            }
        }

        flag = 0;
        for (i = 0; i < n; i++) {
            if (ready[i] == 1) {
                flag = 1;
                break;
            }
        }

        if (flag == 0) {
            pr = -1;
        } else {
            pr = -1;
            for (i = 0; i < n; i++) {
                if (ready[i] == 1) {
                    if (pr == -1 || p[i] < p[pr]) {
                        pr = i;
                    }
                }
            }
        }

        if (pr != pre) {
            if (pr == -1) {
                printf("%d Idle ", op);
            } else {
                printf("%d P%d ", op, pr + 1);
            }
        }

        op++;
```

```
        if (pr != -1) {
            p[pr] = p[pr] - 1;
            if (p[pr] == 0) {
                p[pr] = et[pr];
                ready[pr] = 0;
            }
        }

        pre = pr;
    }

    printf("\n");
}



void prop() {
    // Implementation for proportional share scheduling algorithm
}

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 the number 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];

        for (i = 0; i < n; i++)
            ready[i] = 0;
```

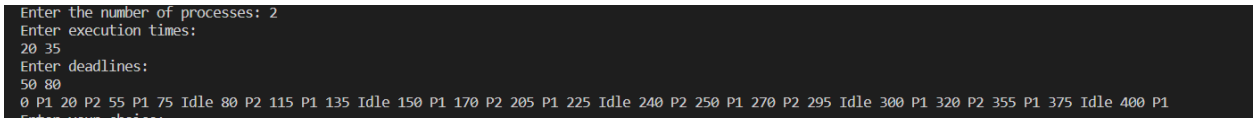
```
        switch (ch) {
            case 1:
                mono();
                break;

            case 2:
                edf();
                break;

            case 3:
                prop();
                break;

            case 4:
                k = 0;
                break;

            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

Output:

Rate Monotonic:



Earliest Deadline First:



Observation:

13-02-23

Write a C program to simulate ⓐ Rate Monotonic ⓑ Earliest-deadline first · ⓒ Proportional Scheduling.

ⓐ
```
#include <stdio.h>
#include <stdlib.h>
int et[10], i, n, dl[10], p[10], ready[10], flag = 1;
int lcm(int a, int b) {
  int max = (a>b) ? a : b;
  while(1) {
      if(max % a == 0 && max % b == 0)
          return max;
      max++;
  }
}
int lcmArray(int arr[], int n) {
    int result = arr[0];
    for(i=1; i<n; i++) {
        result = lcm(result, arr[i]);
    }
    return result;
}
void mono() {
    int time = lcmArray(dl, n);
    int op=0, pr=0, pre = pr;
    while(op <= time) {
        for(i=0; i<n; i++) {
            if(op % dl[i] == 0) {
                ready[i] = 1;
            }
        }
    flag = 0;
    for(i=0; i<n; i++) {
        if(ready[i] == 1) {
            flag = 1;
            break;
        }
    }
    if(flag == 0) {
        pr = -1;
    } else {
        pr = -1;
        for(i=0; i<n; i++) {
            if(ready[i] == 1) {
                if(pr == -1 || dl[i] < dl[pr]) {
                    pr = i;
                }
            }
        }
    }
    if(pr != pre) {
        if(pr == -1) {
            printf("%d Idle ", op);
        } else {
            printf("%d %d ", op, pr+1);
        }
    }
```

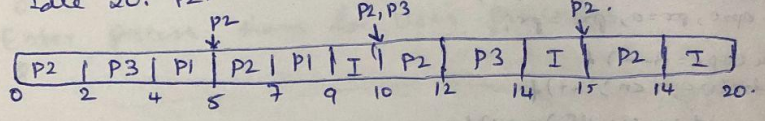
```
        opt++;
        if (pr != -1){
            p[pr] = p[pr] - 1;
            if (p[pr] == 0){
                p[pr] = et[pr];
                ready[pr] = 0;
            }
        }
        pre = pr;
    }
    printf("\n");
}
```

Output:

```
Enter number of processes: 3
Enter execution times:
3 2 2
Enter deadlines:
20 5 10
```

14/2/23

0 P2 2 P3 4 P1 5 P2 7 P1 9 Idle 10 P2 12 P3 14 Idle 15 P2 17 Idle 20. P2.



(b)

```
void edf(){
    int time = lcmArray(dl, n);
    int op=0, pr=0, pre=-1;
    int flag, i;

    while (op <= time){
        for(i=0; i<n; i++){
            if (op%dl[i] == 0){
                read[i] = 1;
        }   }

        flag = 0;
        for(i=0; i<n; i++){
            if (read[i] == 1){
                flag = 1;
                break;
        }   }
        if (flag == 0){
            pr = -1;
        }else{
            pr = -1;
            for(i=0; i<n; i++){
                if (read[i] == 1){
                    if (pr == -1 || p[i] < p[pr]){
                        pr = i;
                    }
        }   }   }
    }
```

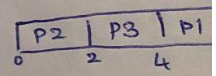
```
        if (pr != pre){
            if (pre == -1){
                printf("
            }else{
                printf("
            }
        }
        op++;
        if (pr != -1){
            p[pr] = p[
            if (p[pr]
                p[
                rea
            }
        }
        pre = pr;
    }
    printf("\n");
}
```

Output:

```
Enter number o
Enter executio
3 2 2
Enter deadline
20 5 10
```

0 P2 2 P3

Idle 20 P2.



```
if (pr != pre) {
    if (pr == -1) {
        printf("%d Idle", op);
    } else {
        printf("%d P%d", op, pr+1);
    }
}
op++;
if (pr != -1) {
    p[pr] = p[pr] - 1;
    if (p[pr] == 0) {
        p[pr] = et[pr];
        ready[pr] = 0;
    }
}
pre = pr;
}
printf("\n");
}
```

15 P2 17

Output:

Enter number of processes: 3.

Enter execution times:

3 2 2

Enter deadline:

20 5 10

0 P2 2 P3 4 P1 7 P2 9 Idle 10 P2 12 P3 14 Idle 15 P2 17 Idle 20 P2.