

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:

```

Enter the number of processes: 3
Enter execution times:
3 2 2
Enter deadlines:
20 5 10
0 P2 2 P3 4 P1 5 P2 7 P1 9 Idle 10 P2 12 P3 14 Idle 15 P2 17 Idle 20 P2

```

Earliest Deadline First:

```

Enter the number of processes: 2
Enter execution times:
20 35
Enter deadlines:
50 80
0 P1 20 P2 55 P1 75 Idle 80 P2 115 P1 135 Idle 150 P1 170 P2 205 P1 225 Idle 240 P2 250 P1 270 P2 295 Idle 300 P1 320 P2 355 P1 375 Idle 400 P1

```

Observation:

```

// C++ program to simulate
// a) Bubble sort
// b) Selection sort
// c) Merge sort

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

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to perform bubble sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
        }
    }
}

// Function to perform selection sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min])
                min = j;
        }
        swap(&arr[i], &arr[min]);
    }
}

// Function to perform merge sort
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l+r)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

// Function to merge two sorted arrays
void merge(int arr[], int l, int m, int r) {
    int n1 = m-l+1;
    int n2 = r-m;
    int* arr1 = new int[n1];
    int* arr2 = new int[n2];
    for (int i = 0; i < n1; i++)
        arr1[i] = arr[l+i];
    for (int i = 0; i < n2; i++)
        arr2[i] = arr[m+1+i];
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j])
            arr[k++] = arr1[i++];
        else
            arr[k++] = arr2[j++];
    }
    while (i < n1)
        arr[k++] = arr1[i++];
    while (j < n2)
        arr[k++] = arr2[j++];
    delete[] arr1;
    delete[] arr2;
}

// Driver code
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    selectionSort(arr, n);
    mergeSort(arr, 0, n-1);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}

```

```

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

```

```

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

```

```

if (rx != -1)
{
    if (arr[rx] == -1)
        printf("Not Ideal\n");
    else
        printf("Not P.T.D", 0, rx + 1);
}

```

```

O(1);
if (rx == -1)
{
    printf("P(0) = P(rx) - 1;
}

```

$P(P(h)) = 0$

$P(h) = ct(h);$

$Nodes(h) = 0; \} 3$

$px = P(h);$

$h = P(h);$

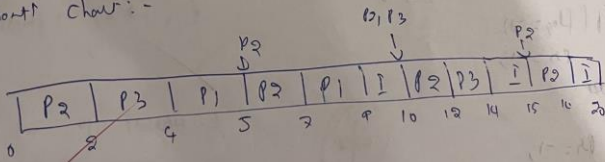
0/P

Enter no of processes: 3

Enter execution time:

~~20 5 10~~

Wait chart:-



b) void rgb()

```
{
    int cur = Ben ans(81, m);
    int op = 0, p = 0, r = -1;
    int flag = 1;
    while (op <= time)
```

```
{
    for (i = 0; i < n; i++)
    {
        if (arr[i] < 0)
            Node[i] = 1;
    }
```

```
    flag = 0;
    for (i = 0; i < n; i++)
```

```
{
    if (Node[i] == 1)
    {
        flag = 1;
        break;
    }
}
```

if (flag == 0)

return -1;

else

{

return -1;

for (i = 0; i < n; i++)

{

if (Node[i] == 1)

{

if (p == -1 || p[i] < p[i+1])

{

p = i; }

}

```

if (P1 < P2)
{
    P1 = P2;
    print("T.D: 01", op);
}
else
{
    print("T.D: P1.D", op, P1);
    op++;
}
if (P1 < -1)
{
    P1 = -1;
}
if (P1 < 0)
{
    P1 = 0;
}
P1 = P1;
print("T.D: 01", op);

```

off

Enter no of processes: 3

Enter execution time:

3 2 2

Enter deadline:

20 5 10

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

10/10

Tracing

19/7/23

