LAB-3

Q. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include<stdio.h>
void sort(int proc_id[],int at[],int bt[],int n)
{
   int temp=0;
   for(int i=0;i<n;i++)
   {
      for(int j=i;j<n;j++)
      {
         if(at[j]<at[i])
         {
            temp=at[i];at[i]=at[j];at[j]=temp;
            temp=bt[j];bt[j]=bt[i];bt[i]=temp;
            temp=proc_id[i];proc_id[i]=proc_id[j];proc_id[j]=temp;
         }
      }
   }
}
void fcfs(int at[],int bt[],int ct[],int tat[],int wt[],int n,int *c)
{
   double ttat=0.0,twt=0.0;
   //completion time
   for(int i=0;i<n;i++)
   {
```

```
        if(*c>=at[i])
            *c+=bt[i];
        else
            *c+=at[i]-ct[i-1]+bt[i];
        ct[i]=*c;
    }
    //turnaround time
    for(int i=0;i<n;i++)
        tat[i]=ct[i]-at[i];
    //waiting time
    for(int i=0;i<n;i++)
        wt[i]=tat[i]-bt[i];
}

void main()
{
    int sn,un,c=0;int n=0;
    printf("Enter number of system processes: ");
    scanf("%d",&sn);n=sn;
    int sproc_id[n],sat[n],sbt[n],sct[n],stat[n],swt[n];
    for(int i=0;i<sn;i++)
        sproc_id[i]=i+1;
    printf("Enter arrival times of the system processes:\n");
    for(int i=0;i<sn;i++)
        scanf("%d",&sat[i]);
    printf("Enter burst times of the system processes:\n");
    for(int i=0;i<sn;i++)
        scanf("%d",&sbt[i]);

    printf("Enter number of user processes: ");
    scanf("%d",&un);n=un;
```

```
    int uproc_id[n],uat[n],ubt[n],uct[n],utat[n],uwt[n];
    for(int i=0;i<un;i++)
        uproc_id[i]=i+1;
    printf("Enter arrival times of the user processes:\n");
    for(int i=0;i<un;i++)
        scanf("%d",&uat[i]);
    printf("Enter burst times of the user processes:\n");
    for(int i=0;i<un;i++)
        scanf("%d",&ubt[i]);

    sort(sproc_id,sat,sbt,sn);
    sort(uproc_id,uat,ubt,un);

    fcfs(sat,sbt,sct,stat,swt,sn,&c);
    fcfs(uat,ubt,uct,utat,uwt,un,&c);

    printf("\nScheduling:\n");
    printf("System processes:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<sn;i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",sproc_id[i],sat[i],sbt[i],sct[i],stat[i],swt[i]);
    printf("User processes:\n");
    for(int i=0;i<un;i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",uproc_id[i],uat[i],ubt[i],uct[i],utat[i],uwt[i]);
}
```

```
Enter number of system processes: 3
Enter arrival times of the system processes:
0
1
4
Enter burst times of the system processes:
3
4
2
Enter number of user processes: 5
Enter arrival times of the user processes:
6
7
8
9
10
Enter burst times of the user processes:
2
3
4
5
6

Scheduling:
System processes:
PID     AT      BT      CT      TAT     WT
1       0       3       3       3       0
2       1       4       7       6       2
3       4       2       9       5       3
User processes:
1       6       2       11      5       3
2       7       3       14      7       4
3       8       4       18      10      6
4       9       5       23      14      9
5       10      6       29      19      13
```

LAB-4

Q.Write a C program to simulate Real-Time CPU Scheduling algorithms:

a)Rate- Monotonic

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

// Function to sort processes based on periods (ascending order)
void sort(int proc[], int b[], int pt[], int n) {
   for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j++) {
         if (pt[j] < pt[i]) {
            // Swapping process ids
            int temp = proc[i];
            proc[i] = proc[j];
            proc[j] = temp;

            // Swapping burst times
            temp = b[i];
            b[i] = b[j];
            b[j] = temp;

            // Swapping periods
            temp = pt[i];
            pt[i] = pt[j];
            pt[j] = temp;
         }
      }
   }
}
```

```
// Function to compute the least common multiple of all periods
int lcmul(int p[], int n) {
    int lcm = p[0];
    for (int i = 1; i < n; i++) {
        int a = lcm, b = p[i];
        while (b != 0) {
            int r = a % b;
            a = b;
            b = r;
        }
        lcm = (lcm * p[i]) / a;
    }
    return lcm;
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], rem[n];

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) scanf("%d", &b[i]);
    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) scanf("%d", &pt[i]);
    for (int i = 0; i < n; i++) proc[i] = i + 1;

    sort(proc, b, pt, n);
    int l = lcmul(pt, n);
```

```
printf("\nRate Monotonic Scheduling:\n");
printf("PID\t Burst\tPeriod\n");
for (int i = 0; i < n; i++) printf("%d\t\t%d\t\t%d\n", proc[i], b[i], pt[i]);

 // Feasibility check
double sum = 0.0;
for (int i = 0; i < n; i++) {
   sum += (double)b[i] / pt[i];
}
double rhs = n * (pow(2.0, (1.0 / n)) - 1.0);
printf("\n%lf <= %lf => %s\n", sum, rhs, (sum <= rhs) ? "true" : "false");
if (sum > rhs) {
   printf("The given set of processes is not schedulable.\n");
   exit(0);
}

printf("Scheduling occurs for %d ms\n\n", l);

int time = 0, prev = -1;
for (int i = 0; i < n; i++) rem[i] = b[i];
int nextRelease[n];
for (int i = 0; i < n; i++) nextRelease[i] = 0;

while (time < l) {
   int taskToExecute = -1;
   for (int i = 0; i < n; i++) {
      if (time == nextRelease[i]) {
         rem[i] = b[i];  // Reset remaining time at the start of the period
         nextRelease[i] += pt[i];  // Schedule next release
      }
      if (rem[i] > 0 && (taskToExecute == -1 || pt[i] < pt[taskToExecute])) {
```

```
        taskToExecute = i;}}
if (taskToExecute != -1) {
    if (prev != taskToExecute) {
        printf("%dms: Task %d is running.\n", time, proc[taskToExecute]);
        prev = taskToExecute;           }
    rem[taskToExecute]--;
 } else if (prev != -1) {
    printf("%dms: CPU is idle.\n", time);
    prev = -1;       }
 time++;    }    return 0;}
```

```
Enter the number of processes: 3
Enter the CPU burst times:
3
2
2
Enter the time periods:
20
5
10

Rate Monotonic Scheduling:
PID      Burst  Period
2               2               5
3               2               10
1               3               20

0.750000 <= 0.779763 => true
Scheduling occurs for 20 ms

0ms: Task 2 is running.
2ms: Task 3 is running.
4ms: Task 1 is running.
5ms: Task 2 is running.
7ms: Task 1 is running.
9ms: CPU is idle.
10ms: Task 2 is running.
12ms: Task 3 is running.
14ms: CPU is idle.
15ms: Task 2 is running.
17ms: CPU is idle.
```

b) Earliest-deadline First

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void
sort (int proc[], int d[], int b[], int pt[], int n)
{
  int temp = 0;
```

```
 for (int i = 0; i < n; i++)
            {
              for (int j = i; j < n; j++)
                        {
                          if (d[j] < d[i])
                                    {
                                      temp = d[j];
                                      d[j] = d[i];
                                      d[i] = temp;
                                      temp = pt[i];
                                      pt[i] = pt[j];
                                      pt[j] = temp;
                                      temp = b[j];
                                      b[j] = b[i];
                                      b[i] = temp;
                                      temp = proc[i];
                                      proc[i] = proc[j];
                                      proc[j] = temp;
                                    }
                        }
            }
}

int
gcd (int a, int b)
{
  int r;
  while (b > 0)
            {
              r = a % b;
              a = b;
```

```
        b = r;
      }
  return a;
}

int
lcmul (int p[], int n)
{
  int lcm = p[0];
  for (int i = 1; i < n; i++)
      {
        lcm = (lcm * p[i]) / gcd (lcm, p[i]);
      }
  return lcm;
}


void
main ()
{
  int n;
  printf ("Enter the number of processes:");
  scanf ("%d", &n);
  int proc[n], b[n], pt[n], d[n], rem[n];
  printf ("Enter the CPU burst times:\n");
  for (int i = 0; i < n; i++)
      {
        scanf ("%d", &b[i]);
        rem[i] = b[i];
      }
  printf ("Enter the deadlines:\n");
```

```
for (int i = 0; i < n; i++)
        scanf ("%d", &d[i]);
printf ("Enter the time periods:\n");
for (int i = 0; i < n; i++)
        scanf ("%d", &pt[i]);
for (int i = 0; i < n; i++)
        proc[i] = i + 1;

sort (proc, d, b, pt, n);
//LCM
int l = lcmul (pt, n);

printf ("\nEarliest Deadline Scheduling:\n");
printf ("PID\t Burst\tDeadline\tPeriod\n");
for (int i = 0; i < n; i++)
        printf ("%d\t\t%d\t\t%d\t\t%d\n", proc[i], b[i], d[i], pt[i]);

printf ("Scheduling occurs for %d ms\n\n", l);

//EDF
int time = 0, prev = 0, x = 0;
int nextDeadlines[n];
for (int i = 0; i < n; i++)
        {
          nextDeadlines[i] = d[i];
          rem[i] = b[i];
        }
while (time < l)
        {
          for (int i = 0; i < n; i++)
                {
```

```
                if (time % pt[i] == 0 && time != 0)
                        {
                          nextDeadlines[i] = time + d[i];
                          rem[i] = b[i];
                        }
                }
        int minDeadline = l + 1;
        int taskToExecute = -1;
        for (int i = 0; i < n; i++)
                {
                 if (rem[i] > 0 && nextDeadlines[i] < minDeadline)
                        {
                          minDeadline = nextDeadlines[i];
                          taskToExecute = i;                    }                    }
        if (taskToExecute != -1)
                {
                 printf ("%dms : Task %d is running.\n", time, proc[taskToExecute]);
                 rem[taskToExecute]--;              }
        else    {
                 printf ("%dms: CPU is idle.\n", time);
                }

        time++;
       }
}
```

```
Enter the number of processes:3
Enter the CPU burst times:
3
2
2
Enter the deadlines:
7
4
8
Enter the time periods:
20
5
10

Earliest Deadline Scheduling:
PID      Burst  Deadline        Period
2               2               4               5
1               3               7               20
3               2               8               10
Scheduling occurs for 20 ms

0ms : Task 2 is running.
1ms : Task 2 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 3 is running.
6ms : Task 3 is running.
7ms : Task 2 is running.
8ms : Task 2 is running.
9ms: CPU is idle.
10ms : Task 2 is running.
11ms : Task 2 is running.
12ms : Task 3 is running.
13ms : Task 3 is running.
14ms: CPU is idle.
15ms : Task 2 is running.
16ms : Task 2 is running.
17ms: CPU is idle.
18ms: CPU is idle.
19ms: CPU is idle.
```

c) Proportional scheduling