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

a) Rate- Monotonic

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void
sort (int proc[], int b[], int pt[], int n)
{
  int temp = 0;
  for (int i = 0; i < n; i++)
          {
            for (int j = i; j < n; j++)
                    {
                      if (pt[j] < pt[i])
                              {
                                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], 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 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 ("LCM=%d\n", l);

  printf ("\nRate Monotone 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]);
```

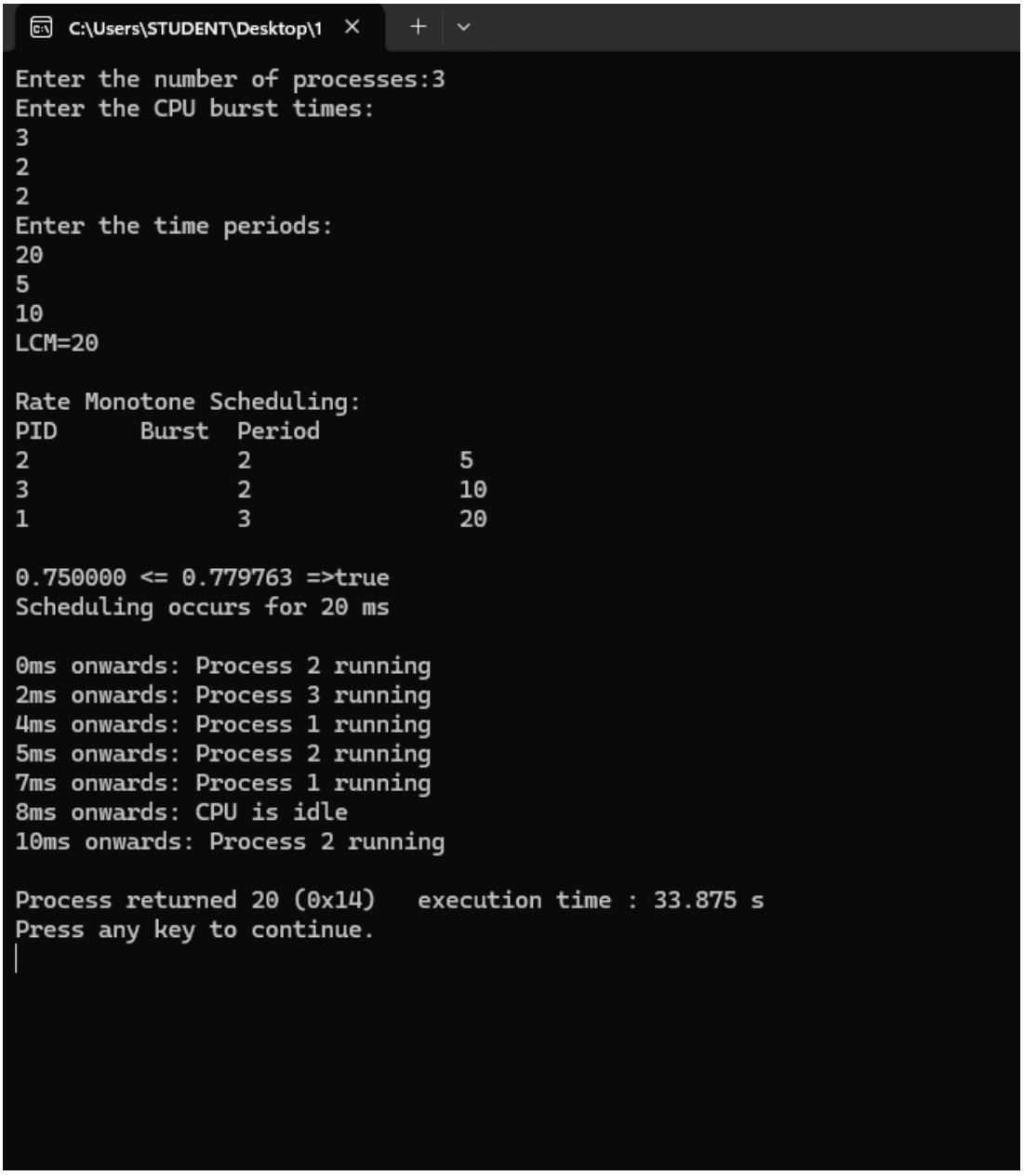
```
double sum = 0.0;
for (int i = 0; i < n; i++)
        {
          sum += (double) b[i] / pt[i];
        }
double rhs = n * (pow (2.0, (1.0 / n)) - 1.0);
printf ("\n%lf <= %lf =>%s\n", sum, rhs, (sum <= rhs) ? "true" : "false");
if (sum > rhs)
        exit (0);

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


int time = 0, prev = 0, x = 0;
while (time < l)
        {
          int f = 0;
          for (int i = 0; i < n; i++)
                  {
                    if (time % pt[i] == 0)
                            rem[i] = b[i];
                    if (rem[i] > 0)
                            {
                              if (prev != proc[i])
                                      {
                                        printf ("%dms onwards: Process %d running\n", time,
                                                        proc[i]);
                                        prev = proc[i];
                                      }
                              rem[i]--;
                              f = 1;
                              break;
                              x = 0;
                            }
                  }
           if (!f)
                  {
                    if (x != 1)
                            {
                              printf ("%dms onwards: CPU is idle\n", time);
                              x = 1;
                            }
```

```
            }
        time++;
      }
}
```

OUTPUT:


C:\Users\STUDENT\Desktop\1
Enter the number of processes:3
Enter the CPU burst times:
3
2
2
Enter the time periods:
20
5
10
LCM=20
Rate Monotone Scheduling:
PID Burst Period
2 2 5
3 2 10
1 3 20
0.750000 <= 0.779763 =>true
Scheduling occurs for 20 ms
0ms onwards: Process 2 running
2ms onwards: Process 3 running
4ms onwards: Process 1 running
5ms onwards: Process 2 running
7ms onwards: Process 1 running
8ms onwards: CPU is idle
10ms onwards: Process 2 running
Process returned 20 (0x14) execution time : 33.875 s
Press any key to continue.

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);

  int l = lcmul (pt, n);

  printf ("\nEarliest Deadline Scheduling:\n");
  printf ("PID\t\t Burst\t\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);


  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++;
          }
}
```

OUTPUT:

```
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.

Process returned 20 (0x14)   execution time : 10.485 s
Press any key to continue.
```

c) Proportional scheduling

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

#define MAX_TASKS 10
#define MAX_TICKETS 100
#define TIME_UNIT_DURATION_MS 100

struct Task {
    int tid;
    int tickets;
};

void schedule(struct Task tasks[], int num_tasks, int *time_span_ms) {
    int total_tickets = 0;

    for (int i = 0; i < num_tasks; i++) {
        total_tickets += tasks[i].tickets;
    }

    srand(time(NULL));

    int current_time = 0;
    int completed_tasks = 0;

        printf("Process Scheduling:\n");

    while (completed_tasks < num_tasks) {
        int winning_ticket = rand() % total_tickets;
        int cumulative_tickets = 0;

        for (int i = 0; i < num_tasks; i++) {
            cumulative_tickets += tasks[i].tickets;

            if (winning_ticket < cumulative_tickets) {
                printf("Time %d-%d: Task %d is running\n", current_time, current_time + 1,
tasks[i].tid);
                current_time++;
                break;
            }
        }
        completed_tasks++;
    }
```

```
    *time_span_ms = current_time * TIME_UNIT_DURATION_MS;
}

int main() {
    struct Task tasks[MAX_TASKS];
    int num_tasks;
    int time_span_ms;

    printf("Enter the number of tasks: ");
    scanf("%d", &num_tasks);

    if (num_tasks <= 0 || num_tasks > MAX_TASKS) {
        printf("Invalid number of tasks. Please enter a number between 1 and %d.\n",
MAX_TASKS);
        return 1;
    }

    printf("Enter number of tickets for each task:\n");
    for (int i = 0; i < num_tasks; i++) {
        tasks[i].tid = i + 1;
        printf("Task %d tickets: ", tasks[i].tid);
        scanf("%d", &tasks[i].tickets);
    }

    printf("\nRunning tasks:\n");
    schedule(tasks, num_tasks, &time_span_ms);

    printf("\nTime span of the Gantt chart: %d milliseconds\n", time_span_ms);

    return 0;
}
```

OUTPUT:

```
Enter the number of tasks: 3
Enter number of tickets for each task:
Task 1 tickets: 10
Task 2 tickets: 20
Task 3 tickets: 30

Running tasks:
Process Scheduling:
Time 0-1: Task 3 is running
Time 1-2: Task 3 is running
Time 2-3: Task 1 is running

Time span of the Gantt chart: 300 milliseconds

Process returned 0 (0x0)   execution time : 7.703 s
Press any key to continue.
```