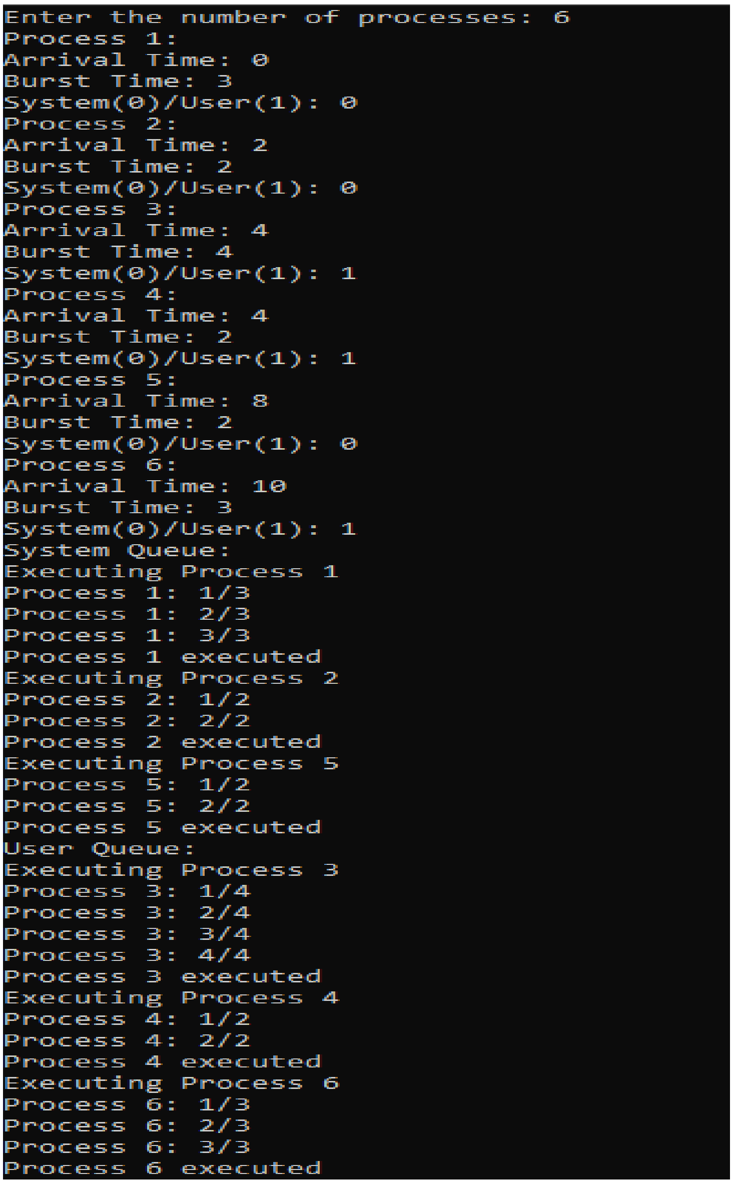
Mutlilevel Queue

Write a C program to simulate a 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>


#define MAX_QUEUE_SIZE 100
typedef struct
{
   int processID;
   int arrivalTime;
   int burstTime;
   int priority;
} Process;
void executeProcess(Process process)
{
   printf("Executing Process %d\n", process.processID);
   for (int i = 1; i <= process.burstTime; i++)
   {
      printf("Process %d: %d/%d\n", process.processID, i, process.burstTime);
   }
   printf("Process %d executed\n", process.processID);
}


void scheduleFCFS(Process queue[], int size)
{
   for (int i = 0; i < size; i++)
   {
      executeProcess(queue[i]);
   }
}
```

```c
int main()
{
    int numProcesses;
    Process processes[MAX_QUEUE_SIZE];
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);
    for (int i = 0; i < numProcesses; i++)
    {
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrivalTime);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burstTime);
        printf("System(0)/User(1): ");
        scanf("%d", &processes[i].priority);
        processes[i].processID = i + 1;
    }
    Process systemQueue[MAX_QUEUE_SIZE];
    int systemQueueSize = 0;
    Process userQueue[MAX_QUEUE_SIZE];
    int userQueueSize = 0;

    for (int i = 0; i < numProcesses; i++)
    {
        if (processes[i].priority == 0)
        {
            systemQueue[systemQueueSize++] = processes[i];
        } else
        {
            userQueue[userQueueSize++] = processes[i];
        }
    }
    printf("System Queue:\n");
    scheduleFCFS(systemQueue, systemQueueSize);
    printf("User Queue:\n");
    scheduleFCFS(userQueue, userQueueSize);


    return 0;
}
```

OUTPUT

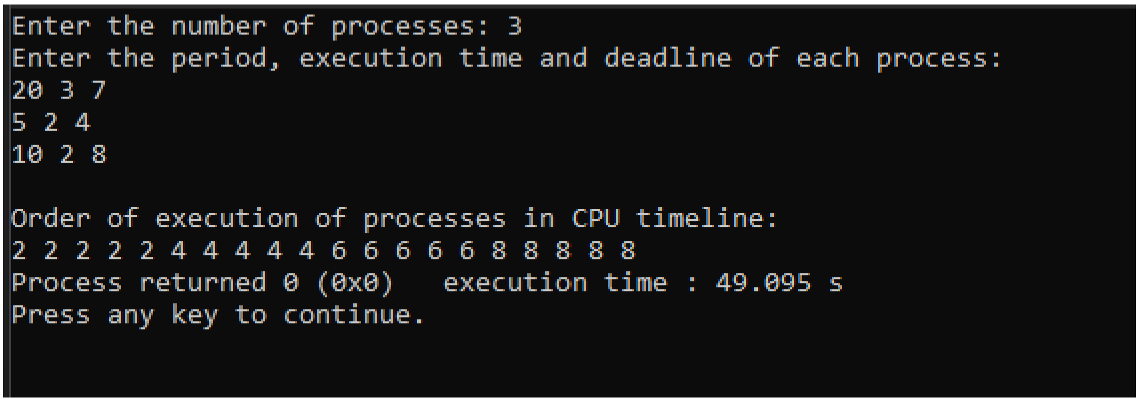


```
Enter the number of processes: 6
Process 1:
Arrival Time: 0
Burst Time: 3
System(0)/User(1): 0
Process 2:
Arrival Time: 2
Burst Time: 2
System(0)/User(1): 0
Process 3:
Arrival Time: 4
Burst Time: 4
System(0)/User(1): 1
Process 4:
Arrival Time: 4
Burst Time: 2
System(0)/User(1): 1
Process 5:
Arrival Time: 8
Burst Time: 2
System(0)/User(1): 0
Process 6:
Arrival Time: 10
Burst Time: 3
System(0)/User(1): 1
System Queue:
Executing Process 1
Process 1: 1/3
Process 1: 2/3
Process 1: 3/3
Process 1 executed
Executing Process 2
Process 2: 1/2
Process 2: 2/2
Process 2 executed
Executing Process 5
Process 5: 1/2
Process 5: 2/2
Process 5 executed
User Queue:
Executing Process 3
Process 3: 1/4
Process 3: 2/4
Process 3: 3/4
Process 3: 4/4
Process 3 executed
Executing Process 4
Process 4: 1/2
Process 4: 2/2
Process 4 executed
Executing Process 6
Process 6: 1/3
Process 6: 2/3
Process 6: 3/3
Process 6 executed
```

3. Simulate Earliest Deadline First for the following and show the order of execution of processes in CPU timeline:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int n;

int period[MAX], execution[MAX], deadline[MAX];

int ready[MAX], task[MAX];

int time = 0;

void swap(int *a, int *b)

{

    int temp = *a;

    *a = *b;

    *b = temp;

}

int gcd(int a, int b)

{

    if (b == 0)

        return a;

    return gcd(b, a % b);

}


void sort()

{

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (deadline[i] > deadline[j])

            {

                swap(&period[i], &period[j]);

                swap(&execution[i], &execution[j]);
                swap(&deadline[i], &deadline[j]);
            }
        }
    }

}
int lcm(int arr[], int n)
{
    int ans = arr[0];
    for (int i = 1; i < n; i++)
        ans = (((arr[i] * ans)) / (gcd(arr[i], ans)));
    return ans;
}
void schedule() {
    int i, j;
for (i = 0; i < n; i++) {
        if (time % period[i] == 0) {
            ready[i] = 1;
        }
    }
    for (i = 0; i < n; i++) {
        if (ready[i] == 1) {
            int min_deadline = 1000000000;
            int min_index = -1;
            for (j = 0; j < n; j++) {
                if (ready[j] == 1 && deadline[j] < min_deadline) {
                    min_deadline = deadline[j];
                    min_index = j;
                }
            }
            task[min_index] += execution[min_index];
            deadline[min_index] += period[min_index];
            ready[min_index] = 0;
        }
    }
}

int main()
{
    int total_time;
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        printf("Enter the period, execution time and deadline of each process:\n");
        for (int i = 0; i < n; i++) {
            scanf("%d %d %d", &period[i], &execution[i], &deadline[i]);
            ready[i] = task[i] = 0;
        }
        sort();
        printf("\nOrder of execution of processes in CPU timeline:\n");
        total_time = lcm(period, n);
        while (time < total_time) {
            schedule();
            printf("%d ", task[0]);
            time++;
        }
        return 0;

}
```

OUTPUT



```
Enter the number of processes: 3
Enter the period, execution time and deadline of each process:
20 3 7
5 2 4
10 2 8

Order of execution of processes in CPU timeline:
2 2 2 2 2 4 4 4 4 4 6 6 6 6 6 8 8 8 8 8
Process returned 0 (0x0)   execution time : 49.095 s
Press any key to continue.
```