Assignment - 4
OPERATING SYSTEM

---

1. 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.

Ans:

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

#define MAX_PROCESSES 100

struct Process {
   int pid;
   int priority;
   int burst_time;
   int arrival_time;
};

struct Queue {
   struct Process *processes[MAX_PROCESSES];
   int front, rear;
};

void initializeQueue(struct Queue *queue) {
   queue->front = -1;
   queue->rear = -1;
}

int isQueueEmpty(struct Queue *queue) {
   return (queue->front == -1);
}

int isQueueFull(struct Queue *queue) {
   return ((queue->rear + 1) % MAX_PROCESSES == queue->front);
}

void enqueue(struct Queue *queue, struct Process *process) {
   if (!isQueueFull(queue)) {
      if (isQueueEmpty(queue))
         queue->front = 0;
      queue->rear = (queue->rear + 1) % MAX_PROCESSES;
      queue->processes[queue->rear] = process;
   } else {
      printf("Queue is full. Cannot enqueue process.\n");
   }
}

struct Process* dequeue(struct Queue *queue) {
   if (!isQueueEmpty(queue)) {
      struct Process *process = queue->processes[queue->front];
      if (queue->front == queue->rear)
```

```
            initializeQueue(queue);
        else
            queue->front = (queue->front + 1) % MAX_PROCESSES;
        return process;
    }
    return NULL;
}

void multiLevelQueueScheduling(struct Process processes[], int n) {
    struct Queue userQueue, systemQueue;
    initializeQueue(&userQueue);
    initializeQueue(&systemQueue);

    for (int i = 0; i < n; i++) {
        if (processes[i].priority == 0)
            enqueue(&userQueue, &processes[i]);
        else
            enqueue(&systemQueue, &processes[i]);
    }

    while (!isQueueEmpty(&systemQueue)) {
        struct Process *process = dequeue(&systemQueue);
        printf("Executing system process with PID %d\n", process->pid);
        // Simulate the execution of the process
    }


    while (!isQueueEmpty(&userQueue)) {
        struct Process *process = dequeue(&userQueue);
        printf("Executing user process with PID %d\n", process->pid);
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[MAX_PROCESSES];

    for (int i = 0; i < n; i++) {
        printf("Enter details for process %d:\n", i + 1);
        processes[i].pid = i + 1;
        printf("Priority (0 for user, 1 for system): ");
        scanf("%d", &processes[i].priority);
        printf("Burst time: ");
        scanf("%d", &processes[i].burst_time);
        printf("Arrival time: ");
        scanf("%d", &processes[i].arrival_time);
    }

    multiLevelQueueScheduling(processes, n);

    return 0;
}
```

```
Arrival time: 1
Enter details for process 2:
Priority (0 for user, 1 for system): 4
Burst time: 3
Arrival time: 2
Enter details for process 3:
Priority (0 for user, 1 for system): 4
Burst time: 6
Arrival time: 2
Enter details for process 4:
Priority (0 for user, 1 for system): 5
Burst time: 5
```