Assignment - 4

OPERATING SYSTEM

TOPIC: Process Scheduling, - PART2

NAME: Devesh Tulshyan
MCA SEM-3
SECTION A
Roll Number: 22

---

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.

   DESCRIPTION
   A multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups based on properties like process type, CPU time, IO access, memory size, etc. In a multi-level queue scheduling algorithm, there will be 'n' number of queues, where 'n' is the number of groups the processes are classified into. Each queue will be assigned a priority and will have its own scheduling algorithm like round-robin scheduling or FCFS. For the process in a queue to execute, all the queues of priority higher than it should be empty, meaning the process in those high-priority queues should have completed its execution. In this scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

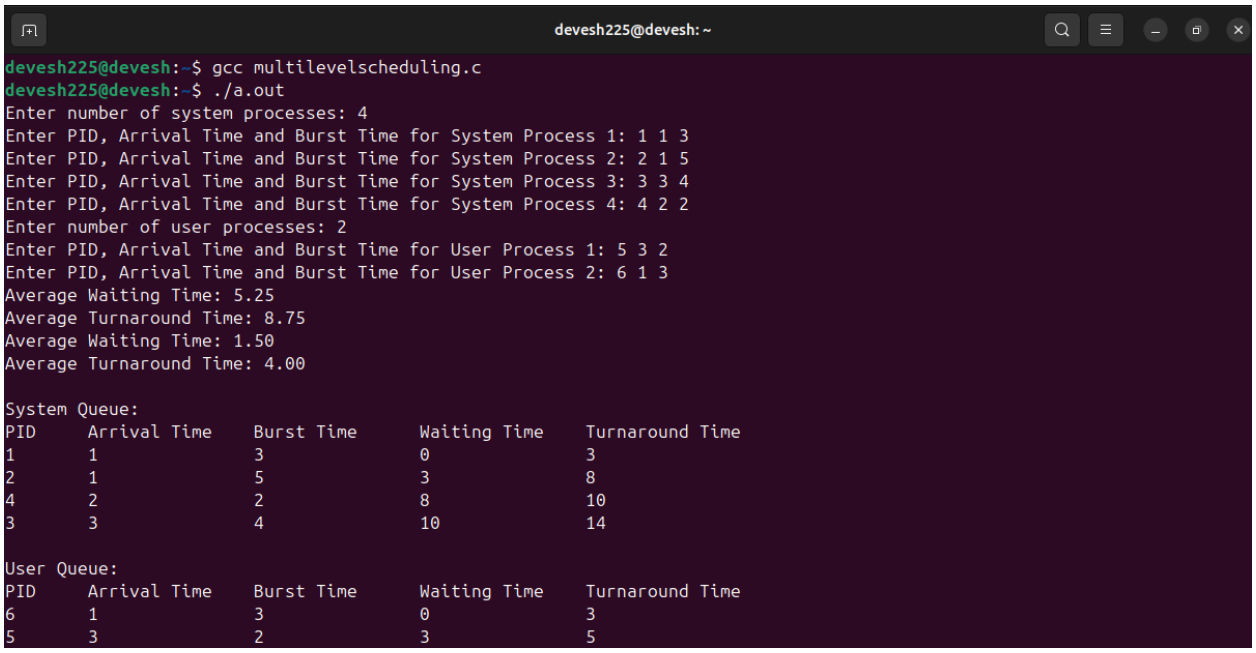
CODE :-

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

#define MAX 10

typedef struct {
   int pid;
   int arrivalTime;
   int burstTime;
```

```
    int waitingTime;
    int turnAroundTime;
} Process;

void calculateTimes(Process queue[], int n) {
    int waitingTime = 0;
    int turnAroundTime = 0;

    for (int i = 0; i < n; i++) {
        if (i == 0) {
            queue[i].waitingTime = 0;
        } else {
            queue[i].waitingTime = queue[i - 1].waitingTime + queue[i - 1].burstTime;
        }
        queue[i].turnAroundTime = queue[i].waitingTime + queue[i].burstTime;
        waitingTime += queue[i].waitingTime;
        turnAroundTime += queue[i].turnAroundTime;
    }

    printf("Average Waiting Time: %.2f\n", (float)waitingTime / n);
    printf("Average Turnaround Time: %.2f\n", (float)turnAroundTime / n);
}

void printQueue(Process queue[], int n, const char* queueName) {
    printf("\n%s:\n", queueName);
    printf("PID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",
               queue[i].pid,
               queue[i].arrivalTime,
               queue[i].burstTime,
               queue[i].waitingTime,
               queue[i].turnAroundTime);
    }
}

int main() {
    int numberSystem, numberUser;
    Process systemQueue[MAX], userQueue[MAX];
    printf("Enter number of system processes: ");
    scanf("%d", &numberSystem);
    for (int i = 0; i < numberSystem; i++) {
        printf("Enter PID, Arrival Time and Burst Time for System Process %d: ", i + 1);
```

```
        scanf("%d %d %d", &systemQueue[i].pid, &systemQueue[i].arrivalTime,
&systemQueue[i].burstTime);
    }
    printf("Enter number of user processes: ");
    scanf("%d", &numberUser);
    for (int i = 0; i < numberUser; i++) {
        printf("Enter PID, Arrival Time and Burst Time for User Process %d: ", i + 1);
        scanf("%d %d %d", &userQueue[i].pid, &userQueue[i].arrivalTime,
&userQueue[i].burstTime);
    }
// system queue scheduling, based on FCFS
    for (int i = 0; i < numberSystem - 1; i++) {
        for (int j = i + 1; j < numberSystem; j++) {
            if (systemQueue[i].arrivalTime > systemQueue[j].arrivalTime) {
                Process temp = systemQueue[i];
                systemQueue[i] = systemQueue[j];
                systemQueue[j] = temp;
            }
        }
    }
// user queue scheduling, based on FCFS
    for (int i = 0; i < numberUser - 1; i++) {
        for (int j = i + 1; j < numberUser; j++) {
            if (userQueue[i].arrivalTime > userQueue[j].arrivalTime) {
                Process temp = userQueue[i];
                userQueue[i] = userQueue[j];
                userQueue[j] = temp;
            }
        }
    }
    calculateTimes(systemQueue, numberSystem);
    calculateTimes(userQueue, numberUser);
    printQueue(systemQueue, numberSystem, "System Queue");
    printQueue(userQueue, numberUser, "User Queue");
    return 0;
}
```

OUTPUT :-


devesh225@devesh: ~
devesh225@devesh:~$ gcc multilevelscheduling.c
devesh225@devesh:~$ ./a.out
Enter number of system processes: 4
Enter PID, Arrival Time and Burst Time for System Process 1: 1 1 3
Enter PID, Arrival Time and Burst Time for System Process 2: 2 1 5
Enter PID, Arrival Time and Burst Time for System Process 3: 3 3 4
Enter PID, Arrival Time and Burst Time for System Process 4: 4 2 2
Enter number of user processes: 2
Enter PID, Arrival Time and Burst Time for User Process 1: 5 3 2
Enter PID, Arrival Time and Burst Time for User Process 2: 6 1 3
Average Waiting Time: 5.25
Average Turnaround Time: 8.75
Average Waiting Time: 1.50
Average Turnaround Time: 4.00
System Queue:
PID Arrival Time Burst Time Waiting Time Turnaround Time
1 1 3 0 3
2 1 5 3 8
4 2 2 8 10
3 3 4 10 14
User Queue:
PID Arrival Time Burst Time Waiting Time Turnaround Time
6 1 3 0 3
5 3 2 3 5