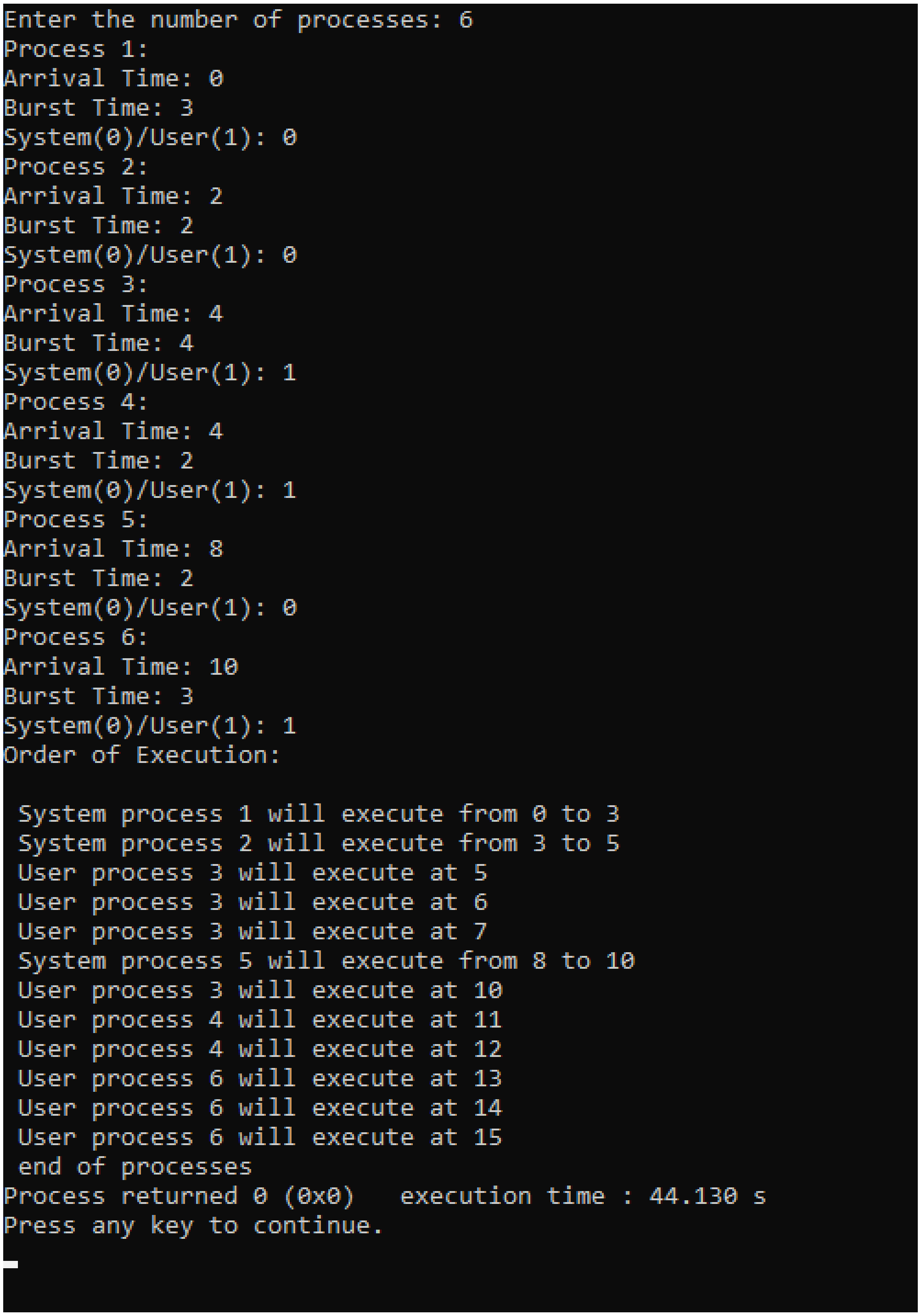
MULTILEVEL SCHEDULING

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 100

int totalTime = 0;
int userProcess = 0, systemProcess = 0;

typedef struct {
    int processID;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int priority; // 0 for system process, 1 for user process
} 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 system[], Process user[]) {
    for (int i = 0; i < systemProcess; i++) {
        for (int j = i + 1; j < systemProcess; j++) {
            if (system[i].arrivalTime > system[j].arrivalTime) {
                Process temp = system[i];
                system[i] = system[j];
            system[j] = temp;
        }
    }
}
for (int i = 0; i < userProcess; i++) {
    for (int j = i + 1; j < userProcess; j++) {
        if (user[i].arrivalTime > user[j].arrivalTime) {
            Process temp = user[i];
            user[i] = user[j];
            user[j] = temp;
        }
    }
}
int completed = 0;
int currentProcess = -1;
int isUserProcess = 0; // Changed bool to int
int size = userProcess + systemProcess;
while (1) {
    int count = 0;
    for (int i = 0; i < systemProcess; i++) {
        if (system[i].remainingTime <= 0) {
            count++;
        }
    }
    for (int j = 0; j < userProcess; j++) {
        if (user[j].remainingTime <= 0) {
            count++;
        }
    }
    if (count == size) {
        printf("\n end of processes");
        exit(0);
    }
    for (int i = 0; i < systemProcess; i++) {
            if (totalTime >= system[i].arrivalTime && system[i].remainingTime >
0) {
                currentProcess = i;
                isUserProcess = 0; // Changed true to 0
                break;
            }
        }
        if (currentProcess == -1) {
            for (int j = 0; j < userProcess; j++) {
                if (totalTime >= user[j].arrivalTime && user[j].remainingTime > 0) {
                    currentProcess = j;
                    isUserProcess = 1; // Changed true to 1
                    break;
                }
            }
        }
        if (currentProcess == -1) {
            totalTime++;
            printf("\n %d   idle time...", totalTime);
            if (totalTime == 1000) {
                exit(0);
            }
            continue;
        }
        if (isUserProcess == 1) { // Changed true to 1
            user[currentProcess].remainingTime--;
            printf("\n User process %d will execute at %d ",
user[currentProcess].processID, (totalTime));
            totalTime++;
            isUserProcess = 0; // Changed true to 0
            currentProcess = -1;
            if (user[currentProcess].remainingTime == 0) {
                completed++;
            }
        } else {
            int temp = totalTime;
            while (system[currentProcess].remainingTime--) {
                totalTime++;
            }
            if (system[currentProcess].remainingTime == 0) {
                completed++;
            }
            printf("\n System process %d will execute from %d to %d ",
system[currentProcess].processID, temp, (totalTime));
            isUserProcess = 0; // Changed true to 0
            currentProcess = -1;
        }
    }
}

int main() {
    int numProcesses;
    Process processes[MAX_QUEUE_SIZE];

    // Reading the number of processes
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);
    // Reading process details
    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;
        processes[i].remainingTime = processes[i].burstTime;
        if (processes[i].priority == 1) {
            userProcess++;
        } else {
            systemProcess++;
        }
    }

    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("Order of Execution:\n");
    scheduleFCFS(systemQueue, userQueue);
    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
Order of Execution:

 System process 1 will execute from 0 to 3
 System process 2 will execute from 3 to 5
 User process 3 will execute at 5
 User process 3 will execute at 6
 User process 3 will execute at 7
 System process 5 will execute from 8 to 10
 User process 3 will execute at 10
 User process 4 will execute at 11
 User process 4 will execute at 12
 User process 6 will execute at 13
 User process 6 will execute at 14
 User process 6 will execute at 15
 end of processes
Process returned 0 (0x0)   execution time : 44.130 s
Press any key to continue.
```