OS LAB 3

AIM: Write a C program to simulate 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.

SOURCE CODE

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

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

void sort_by_arrival_time(struct Process* processes, int num_processes) {
   struct Process temp;
   for (int i = 0; i < num_processes - 1; i++) {
      for (int j = i + 1; j < num_processes; j++) {
         if (processes[i].arrival_time > processes[j].arrival_time) {
            temp = processes[i];
            processes[i] = processes[j];
            processes[j] = temp;
         }
      }
   }
}

void multi_level_queue_scheduling(struct Process* processes, int num_processes) {
```

```
int current_time = 0;



int system_count = 0;
int user_count = 0;
for (int i = 0; i < num_processes; i++) {
    if (processes[i].priority == 1) {
        system_count++;
    } else {
        user_count++;
    }
}

// Allocate memory for system and user process queues
struct Process* system_queue = (struct Process*)malloc(system_count * sizeof(struct Process));
struct Process* user_queue = (struct Process*)malloc(user_count * sizeof(struct Process));



int system_idx = 0;
int user_idx = 0;
for (int i = 0; i < num_processes; i++) {
    if (processes[i].priority == 1) {
        system_queue[system_idx++] = processes[i];
    } else {
        user_queue[user_idx++] = processes[i];
    }
}



sort_by_arrival_time(system_queue, system_count);
sort_by_arrival_time(user_queue, user_count);
```

```
    printf("Simulation Result:\n");


    for (int i = 0; i < system_count; i++) {
        struct Process* current_process = &system_queue[i];
        printf("Time %d: Process %d (System) is running\n", current_time, current_process->pid);
        current_time += current_process->burst_time;
        printf("Time %d: Process %d (System) is completed\n", current_time, current_process->pid);
    }


    for (int i = 0; i < user_count; i++) {
        struct Process* current_process = &user_queue[i];
        printf("Time %d: Process %d (User) is running\n", current_time, current_process->pid);
        current_time += current_process->burst_time;
        printf("Time %d: Process %d (User) is completed\n", current_time, current_process->pid);
    }

    free(system_queue);
    free(user_queue);
}

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

    struct Process* processes = (struct Process*)malloc(num_processes * sizeof(struct Process));

    for (int i = 0; i < num_processes; i++) {
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
```

```
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        printf("Enter priority for process %d (1 for System, 0 for User): ", i + 1);
        scanf("%d", &processes[i].priority);
        processes[i].pid = i + 1;
    }

    multi_level_queue_scheduling(processes, num_processes);

    free(processes);

    return 0;
}
```

OUTPUT SCREENSHOTS