Lab no 3

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 RR and

FCFS scheduling for the processes in each queue.

#include <stdio.h>

struct Process {

    int id, burst\_time, arrival\_time, queue;

    int waiting\_time, turnaround\_time, response\_time;

};

void round\_robin(struct Process p[], int n, int quantum) {

    int remaining\_time[n], completed = 0, time = 0;

    for (int i = 0; i < n; i++) remaining\_time[i] = p[i].burst\_time;

    while (completed < n) {

        for (int i = 0; i < n; i++) {

            if (remaining\_time[i] > 0) {

                if (remaining\_time[i] > quantum) {

                    time += quantum;

                    remaining\_time[i] -= quantum;

                } else {

                    time += remaining\_time[i];

                    p[i].waiting\_time = time - p[i].arrival\_time - p[i].burst\_time;

                    p[i].turnaround\_time = time - p[i].arrival\_time;

                    p[i].response\_time = p[i].waiting\_time;

                    remaining\_time[i] = 0;

                    completed++;

                }

            }

        }

    }

}

void fcfs(struct Process p[], int n, int start\_time) {

    int time = start\_time;

    for (int i = 0; i < n; i++) {

        if (time < p[i].arrival\_time)

            time = p[i].arrival\_time;

        p[i].waiting\_time = time - p[i].arrival\_time;

        p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time;

        p[i].response\_time = p[i].waiting\_time;

        time += p[i].burst\_time;

    }

}

int main() {

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    struct Process processes[n], system\_queue[n], user\_queue[n];

    int sys\_count = 0, user\_count = 0;

    printf("Enter Burst Time, Arrival Time and Queue of each process: \n");

    for (int i = 0; i < n; i++) {

        printf("P%d: ", i + 1);

        scanf("%d %d %d", &processes[i].burst\_time, &processes[i].arrival\_time, &processes[i].queue);

        processes[i].id = i + 1;

        if (processes[i].queue == 1)

            system\_queue[sys\_count++] = processes[i];

        else if (processes[i].queue == 2)

            user\_queue[user\_count++] = processes[i];

    }

    int quantum = 2;

    round\_robin(system\_queue, sys\_count, quantum);

    int last\_exec\_time = (sys\_count > 0) ? system\_queue[sys\_count - 1].turnaround\_time : 0;

    fcfs(user\_queue, user\_count, last\_exec\_time);

    printf("\nProcess\tWT\tTAT\tRt\n");

    for (int i = 0; i < sys\_count; i++)

        printf("P%d\t%d\t%d\t%d\n", system\_queue[i].id, system\_queue[i].waiting\_time, system\_queue[i].turnaround\_time, system\_queue[i].response\_time);

    for (int i = 0; i < user\_count; i++)

        printf("P%d\t%d\t%d\t%d\n", user\_queue[i].id, user\_queue[i].waiting\_time, user\_queue[i].turnaround\_time, user\_queue[i].response\_time);

    float avg\_wait = 0, avg\_tat = 0, avg\_resp = 0;

    for (int i = 0; i < sys\_count; i++) {

        avg\_wait += system\_queue[i].waiting\_time;

        avg\_tat += system\_queue[i].turnaround\_time;

        avg\_resp += system\_queue[i].response\_time;

    }

    for (int i = 0; i < user\_count; i++) {

        avg\_wait += user\_queue[i].waiting\_time;

        avg\_tat += user\_queue[i].turnaround\_time;

        avg\_resp += user\_queue[i].response\_time;

    }

    int total = sys\_count + user\_count;

    printf("\nAverage Waiting Time: %.2f", avg\_wait / total);

    printf("\nAverage Turn Around Time: %.2f", avg\_tat / total);

    printf("\nAverage Response Time: %.2f", avg\_resp / total);

    printf("\nThroughput: %.2f\n", (float)total / avg\_tat \* total);

    return 0;

}

