**Week - 4**

**(13 July 2023)**

**Experiment - 4**

**Question:**

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.

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct process {

int pid;

int arrival\_time;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

void FCFS(struct process \*queue, int n) {

int i, j;

struct process temp;

for (i = 0; i < n; i++) {

for (j = i + 1; j < n; j++) {

if (queue[i].arrival\_time > queue[j].arrival\_time) {

temp = queue[i];

queue[i] = queue[j];

queue[j] = temp;

}

}

}

}

int main() {

int n, i;

struct process \*system\_queue, \*user\_queue;

int system\_n = 0, user\_n = 0;

float avg\_waiting\_time = 0, avg\_turnaround\_time = 0;

printf("Enter the number of processes: ");

scanf("%d", &n);

system\_queue = (struct process \*) malloc(n \* sizeof(struct process));

user\_queue = (struct process \*) malloc(n \* sizeof(struct process));

for (i = 0; i < n; i++) {

struct process p;

printf("Enter arrival time, burst time, and priority (0-System/1-User) for process %d: ", i + 1);

scanf("%d %d %d", &p.arrival\_time, &p.burst\_time, &p.priority);

p.pid = i + 1;

p.waiting\_time = 0;

p.turnaround\_time = 0;

if (p.priority == 0) {

system\_queue[system\_n++] = p;

} else {

user\_queue[user\_n++] = p;

}

}

FCFS(system\_queue, system\_n);

FCFS(user\_queue, user\_n);

int time = 0;

int s=0,u=0;

while(s<system\_n || u<user\_n){

if(system\_queue[s].arrival\_time <= time){

if(user\_queue[u].arrival\_time <= time && user\_queue[u].arrival\_time < system\_queue[s].arrival\_time){

user\_queue[u].waiting\_time = time - user\_queue[u].arrival\_time;

time += user\_queue[u].burst\_time;

user\_queue[u].turnaround\_time = user\_queue[u].waiting\_time + user\_queue[u].burst\_time;

avg\_waiting\_time += user\_queue[u].waiting\_time;

avg\_turnaround\_time += user\_queue[u].turnaround\_time;

u++;

}

else{

system\_queue[s].waiting\_time = time - system\_queue[s].arrival\_time;

time += system\_queue[s].burst\_time;

system\_queue[s].turnaround\_time = system\_queue[s].waiting\_time + system\_queue[s].burst\_time;

avg\_waiting\_time += system\_queue[s].waiting\_time;

avg\_turnaround\_time += system\_queue[s].turnaround\_time;

s++;

}

}

else if(user\_queue[u].arrival\_time <= time){

user\_queue[u].waiting\_time = time - user\_queue[u].arrival\_time;

time += user\_queue[u].burst\_time;

user\_queue[u].turnaround\_time = user\_queue[u].waiting\_time + user\_queue[u].burst\_time;

avg\_waiting\_time += user\_queue[u].waiting\_time;

avg\_turnaround\_time += user\_queue[u].turnaround\_time;

u++;

}

else{

if(system\_queue[s].arrival\_time <= user\_queue[u].arrival\_time){

time = system\_queue[s].arrival\_time;

}

else{

time = user\_queue[u].arrival\_time;

}

}

}

avg\_waiting\_time /= n;

avg\_turnaround\_time /= n;

printf("PID\tBurst Time\tPriority\tQueue Type\tWaiting Time\tTurnaround Time\n");

for (i = 0; i < system\_n; i++) {

printf("%d\t%d\t\t%d\t\tSystem\t\t%d\t\t%d\n", system\_queue[i].pid, system\_queue[i].burst\_time, system\_queue[i].priority, system\_queue[i].waiting\_time, system\_queue[i].turnaround\_time);

}

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

printf("%d\t%d\t\t%d\t\tUser\t\t%d\t\t%d\n", user\_queue[i].pid, user\_queue[i].burst\_time, user\_queue[i].priority, user\_queue[i].waiting\_time, user\_queue[i].turnaround\_time);

}

printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

printf("Average Turnaround Time: %.2f\n", avg\_turnaround\_time);

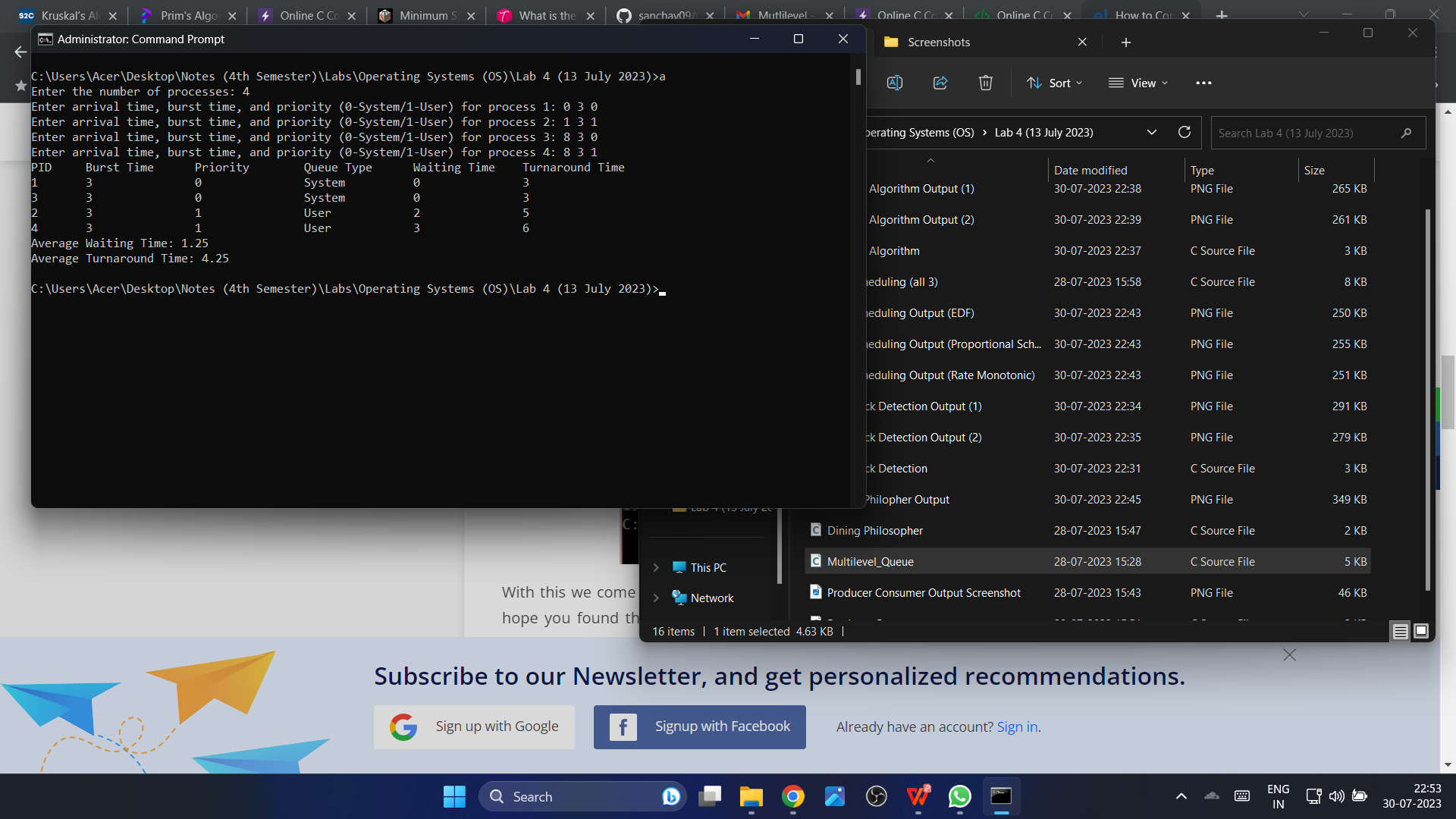
free(system\_queue);

free(user\_queue);

return 0;

}

**Output:**



**Observation Book Pictures:**

