WEEK3

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.

1.1.1 Code:

```
#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

```
Enter the number of processes: 4
Enter arrival time, burst time, and priority (0-System/1-User) for process 1: 0 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 2: 1 3 1
Enter arrival time, burst time, and priority (0-System/1-User) for process 3: 8 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 4: 8 3 1
PID     Burst Time      Priority        Queue Type      Waiting Time    Turnaround Time
1       3               0               System          0               3
3       3               0               System          0               3
2       3               1               User            2               5
4       3               1               User            3               6
Average Waiting Time: 1.25
Average Turnaround Time: 4.25
```

Lab 6 - A

1) write a C prog to simulate multi-level queue scheduling algo considering the following scenarios. All the process in the system are divided into 2 categories system process & user process. System process to be given higher priority than user

code :-

```
#include<stdio.h>
int sptat[10], upar[10], u, n, n2p[10], p2[10]
int sptat[10], uptat[10] time>0, op=0, 4, 2, pt;
int spar[10], uptat[10], uptat[10], sptat[10]
float spatat=[0] isp[0] upatat=0 u;
void process (int x, int us system) {
    if (is system) {
        op += sppt[x];
        sptat[x] = op - spat[x];
        sppt[x] = 0;
        spat[x] = sptat[x] - pt[x];
        spatat = sptat[x];
        spwat += spat[x];
    }
    else
    {
        opt = uppt[x];
        uptat[x] = op - upat[x];
        uppt[x] = 0;
        upat[x] = uptat - pa[x];
        upatat += uptat[x];
    } upwat += upat[x];
    }
int main() {
    printf("Enter the no of system processes:");
    scanf("%d", &n)
    printf("Enter the no of user processes:\n");
    scanf("%d", &n);
```

```
printf("Enter arrival time for system processes");
for(i=0; i<n1; i++)
printf("Enter burst time for system process:\n");
for(i=0; i<n1; i++)
scanf("%d", &spt[i]);
printf("Enter the arrival and burst time for user processes");
for(i=0; i<n2; i++){
scanf("%d", &upt[i]);
scanf("%d", &upt[i]);
}
for(i=0; i<n1; i++){
time += spd[i];
p1[i] = spt[i];
}
for(i=0; i<n2; i++){
time += upt[i];
p2[i] = upt[i];
}
printf("\n");
while(op<time){
y = -1;
z = -1;
for(i=0; i<m; i++){
if(op>=spd[i] && spt[i]!=0)
{
y = i
break;
}
}
for(i>0; i<n2; i++){
if(op>=upd[i] && upt[i]!=0){
z = i;
break;
}
}
```

```
if (y != -1){
printf ("%d %d", op, y, i+1);
process (y, i);
}
else if (z != -1){
printf ("%d %d", op, z+1);
process (z, 0);
}
else {
op++;
}
}
printf ("\n");
printf ("system process: \n");
for (i=0; i<n1; i++)
printf ("%d %d %d %d %d\n", i+1, sptat[i]);
printf ("Average turnaround time system process);
%.2f \n", sptat/n1);
printf ("Average waiting time system process: \n");
printf ("\n")
printf ("user process: \n");
for (i>0; i<n2; i++)
printf ("UR %d %d %d %d \n", i+, uptat[i], uptat[i]);
printf ("Average turnaround time user process: %.2f \n",
upstat/n2);
printf ("Average waiting time user process); %.2f
upwat/n2);
return 0;
}
```

output

Enter no of system process : 3

enter no of user process : 1

enter arrival time of system process : 0 0 1

enter burst time of system process : 4 3 5

enter arrival burst time of user process : 0 7

System	Process	
SP1	4	6
SP2	7	4
SP3	10	5

Average Turnaround time (System process) : 7

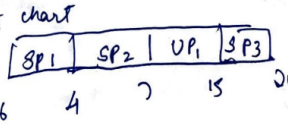
Average waiting Time (System process) : 3

Average Turnaround Time (user process) : 15

Average waiting time (user process) : 7

Tracing

Gant chart



Turn Around Time = CT - PT

UPT = 15 - 0 = 15

Waiting Time = TAT - BT

UP1 = 15 - 8 = 7

TAT

SP1 = 4 - 4 = 0

SP2 = 7 - 3 = 4

SP3 = 20 - 10 = 10

$SP_1 = C_1 - 0 = C_1$

$SP_2 = 7 - 4 = 3$

$SP_3 = 10 - 5 = 5$

Average TAT = 7

Average WT = 3

8/10

19/7/23