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;
```

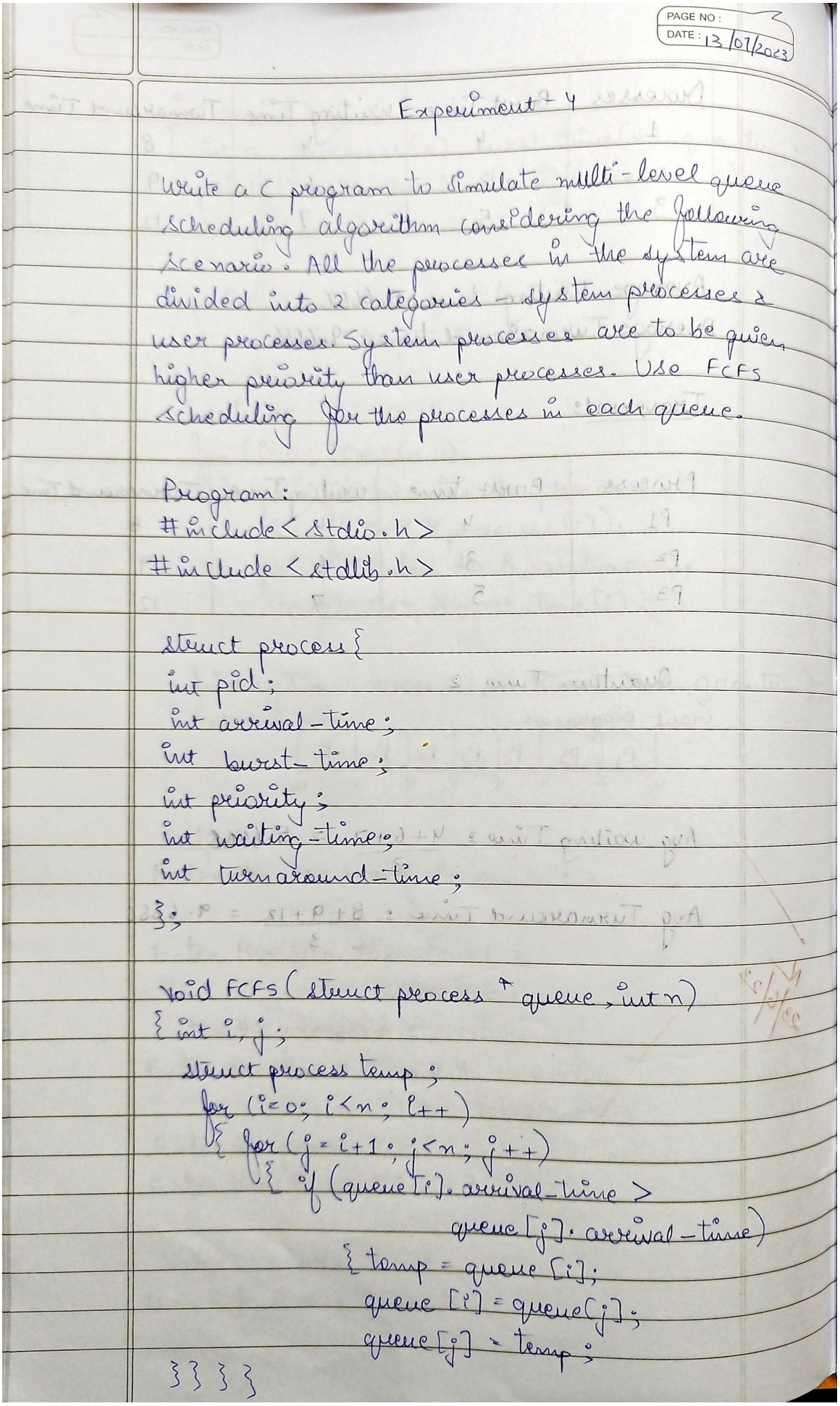
```
        p.waiting_time = 0;
        p.turnaround_time = 0;
        if (p.priority == 0) {
            system_queue[system_n++] = p;
        } else {
            user_queue[user_n++] = p;
        }
    }

    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:

```
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
```

Observation Book Pictures:



PAGE NO :
DATE : 13/07/2023

Experiment - 4

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario: All the processes in the system are divided into 2 categories - system processes & 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;
}}}}
```

PAGE NO :
DATE :

```
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].waiting_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\t Burst Time\t Priority\t Queue Type\t
        Waiting Time\t Turnaround Time\n");
for(i=0; i< system_n; i++)
   { printf("%d\t %d\t\t%d\t\t System\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 \t User \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: %f \n",
            avg_waiting_time);          avg_turnaround_time
    printf("Average Turnaround Time: %f \n");

    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

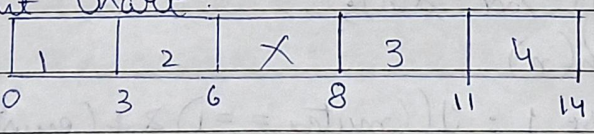
PID	Burst time	Priority	Queue Type	Waiting Time	Turnaround_Time
1	3	0	System	0	3
3	3	0	System	0	3
2	3	1	User	2	5
4	3	1	User	3	6

Average waiting time = 1.25
Average Turnaround time = 4.25.

Traversal:

Process	System-0/user-1	arrival time	B.T	W.T	TAT
1	0	0	3	0	3
2	1	1	3	2	5
3	0	8	3	0	3
4	1	8	3	3	6

Grant Chart:



10/10

N 20/7/23