Multi-Level Queue Scheduling

Perumalla Dharan
AP21110010201

1. Implement Multi-Level Queue scheduling.
   Read the details of N processes, like Process ID, Type of Process, Burst Time, Priority (If the process is a student process), etc. Find the completion time, Turn around, and wait time of each process and the average of these.
   Print the Gant chart also.
   1. System process: FCFS
   2. Interactive processes: Round Robin with TQ: 2
   3. Interactive editing processes: Round Robin with TQ: 4
   4. Batch processes: SJF
   5. Student processes: Priority scheduling

CODE

```
// Implement multilevel queue scheduling

// Read the details of n processes, like process ID,
Type of process, Burst time, Priority
// (If the process is student process), etc. Find the
completion time,
// turn around time, wait time of each of the pricesses
and also find the averages.

// Arrival time is 0 for all the processes.

// Print the gantt chart.
```

```
// 1. System Processes                    :
FCFS
// 2. Interactive Processes               :
Round Robin with TQ=2
// 3. Interactive editing Processes       :
Round Robin with TQ=4
// 4. Batch Processes                     :          SJF
// 5. Student Processes                   :
Priority Scheduling

#include <iostream>
using namespace std;

// A structure to represent a process
struct Process
{
    int pid;
    int type;
    int burst_time;
    int priority;
    int completion_time;
    int turn_around_time;
    int wait_time;
    int remaining_time;
    int response_time;
};

// Structure for FCFS
struct fcfs{
    int pid;
    int burst_time;
    int completion_time;
    int turn_around_time;
```

```
    int wait_time;
    int remaining_time;
    int response_time;
};

// Structure for Round Robin time quantum 2
struct rr2{
    int pid;
    int burst_time;
    int completion_time;
    int turn_around_time;
    int wait_time;
    int remaining_time;
    int response_time;
};

// Structure for Round Robin time quantum 4
struct rr4{
    int pid;
    int burst_time;
    int completion_time;
    int turn_around_time;
    int wait_time;
    int remaining_time;
    int response_time;
};

// Structure for SJF
struct sjf{
    int pid;
    int burst_time;
    int completion_time;
    int turn_around_time;
```

```
    int wait_time;
    int remaining_time;
    int response_time;
};

// Structure for Priority
struct priority{
    int pid;
    int burst_time;
    int priority;
    int completion_time;
    int turn_around_time;
    int wait_time;
    int remaining_time;
    int response_time;
};

// Function to find the waiting time for fcfs
int dofcfs(fcfs f[], int fcfs_count,int time){
    for (int i = 0; i < fcfs_count; i++)
    {
        f[i].completion_time=0;
        f[i].turn_around_time=0;
        f[i].wait_time=0;
        f[i].response_time=0;
        f[i].completion_time=time+f[i].burst_time;
        time=f[i].completion_time;
        f[i].turn_around_time=f[i].completion_time;

f[i].wait_time=f[i].turn_around_time-f[i].burst_time;
        f[i].response_time=f[i].wait_time;
    }
```

```
    // cout<<"FCFS"<<endl;
    // cout<<"PID\tBurst Time\tCompletion Time\tTurn
Around Time\tWait Time\tResponse Time"<<endl;
    for (int i = 0; i < fcfs_count; i++)
    {

cout<<f[i].pid<<"\t"<<f[i].burst_time<<"\t\t"<<f[i].comp
letion_time<<"\t\t"<<f[i].turn_around_time<<"\t\t\t"<<f[
i].wait_time<<"\t\t"<<endl;
    }
    return time;
}

// Function to find the waiting time for Round Robin
time quantum 2
int dorr2(rr2 r2[], int rr2_count,int time){
    for (int i = 0; i < rr2_count; i++)
    {
        r2[i].completion_time=0;
        r2[i].turn_around_time=0;
        r2[i].wait_time=0;
        r2[i].response_time=0;
        r2[i].remaining_time=r2[i].burst_time;
    }
    int flag=0;
    while (flag==0)
    {
        flag=1;
        for (int i = 0; i < rr2_count; i++)
        {
            if (r2[i].remaining_time>2)
            {
                time+=2;
```

```
                r2[i].remaining_time-=2;
                flag=0;
            }
            else if (r2[i].remaining_time>0)
            {
                time+=r2[i].remaining_time;
                r2[i].remaining_time=0;
                r2[i].completion_time=time;

r2[i].turn_around_time=r2[i].completion_time;

r2[i].wait_time=r2[i].turn_around_time-r2[i].burst_time;
                r2[i].response_time=r2[i].wait_time;
            }
        }
    }
    // cout<<"Round Robin Time Quantum 2"<<endl;
    // cout<<"PID\tBurst Time\tCompletion Time\tTurn
Around Time\tWait Time\tResponse Time"<<endl;
    for (int i = 0; i < rr2_count; i++)
    {

cout<<r2[i].pid<<"\t"<<r2[i].burst_time<<"\t\t"<<r2[i].c
ompletion_time<<"\t\t"<<r2[i].turn_around_time<<"\t\t\t"
<<r2[i].wait_time<<"\t\t"<<endl;
    }
    return time;
}

// Function to find the waiting time for Round Robin
time quantum 4
int dorr4(rr4 r4[], int rr4_count, int time){
    for (int i = 0; i < rr4_count; i++)
```

```
    {
        r4[i].completion_time=0;
        r4[i].turn_around_time=0;
        r4[i].wait_time=0;
        r4[i].response_time=0;
        r4[i].remaining_time=r4[i].burst_time;
    }
    int flag=0;
    while (flag==0)
    {
        flag=1;
        for (int i = 0; i < rr4_count; i++)
        {
            if (r4[i].remaining_time>4)
            {
                time+=4;
                r4[i].remaining_time-=4;
                flag=0;
            }
            else if (r4[i].remaining_time>0)
            {
                time+=r4[i].remaining_time;
                r4[i].remaining_time=0;
                r4[i].completion_time=time;

r4[i].turn_around_time=r4[i].completion_time;

r4[i].wait_time=r4[i].turn_around_time-r4[i].burst_time;
                r4[i].response_time=r4[i].wait_time;
            }
        }
    }
    // cout<<"Round Robin Time Quantum 4"<<endl;
```

```
    // cout<<"PID\tBurst Time\tCompletion Time\tTurn
Around Time\tWait Time\tResponse Time"<<endl;
    for (int i = 0; i < rr4_count; i++)
    {

cout<<r4[i].pid<<"\t"<<r4[i].burst_time<<"\t\t"<<r4[i].c
ompletion_time<<"\t\t"<<r4[i].turn_around_time<<"\t\t\t"
<<r4[i].wait_time<<"\t\t"<<endl;
    }
    return time;
}

// Function to find the waiting time for SJF
int dosjf(sjf s[], int sjf_count, int time){
    for (int i = 0; i < sjf_count; i++)
    {
        s[i].completion_time=0;
        s[i].turn_around_time=0;
        s[i].wait_time=0;
        s[i].response_time=0;
        s[i].remaining_time=s[i].burst_time;
    }
    int flag=0;
    while (flag==0)
    {
        flag=1;
        for (int i = 0; i < sjf_count; i++)
        {
            if (s[i].remaining_time>0)
            {
                time+=s[i].remaining_time;
                s[i].remaining_time=0;
                s[i].completion_time=time;
```

```
s[i].turn_around_time=s[i].completion_time;

s[i].wait_time=s[i].turn_around_time-s[i].burst_time;
                s[i].response_time=s[i].wait_time;
            }
        }
    }
    // cout<<"Shortest Job First"<<endl;
    // cout<<"PID\tBurst Time\tCompletion Time\tTurn
Around Time\tWait Time\tResponse Time"<<endl;
    for (int i = 0; i < sjf_count; i++)
    {

cout<<s[i].pid<<"\t"<<s[i].burst_time<<"\t\t"<<s[i].comp
letion_time<<"\t\t"<<s[i].turn_around_time<<"\t\t\t"<<s[
i].wait_time<<"\t\t"<<endl;
    }
    return time;
}

// Function to find the waiting time for Priority
int dopriority(priority p[], int priority_count,int
time){
    for (int i = 0; i < priority_count; i++)
    {
        p[i].completion_time=0;
        p[i].turn_around_time=0;
        p[i].wait_time=0;
        p[i].response_time=0;
        p[i].remaining_time=p[i].burst_time;
    }
    int flag=0;
```

```
    while (flag==0)
    {
        flag=1;
        for (int i = 0; i < priority_count; i++)
        {
            if (p[i].remaining_time>0)
            {
                time+=p[i].remaining_time;
                p[i].remaining_time=0;
                p[i].completion_time=time;

p[i].turn_around_time=p[i].completion_time;

p[i].wait_time=p[i].turn_around_time-p[i].burst_time;
                p[i].response_time=p[i].wait_time;
            }
        }
    }
    // cout<<"Priority"<<endl;
    // cout<<"PID\tBurst Time\tCompletion Time\tTurn
Around Time\tWait Time\tResponse Time"<<endl;
    for (int i = 0; i < priority_count; i++)
    {

cout<<p[i].pid<<"\t"<<p[i].burst_time<<"\t\t"<<p[i].comp
letion_time<<"\t\t"<<p[i].turn_around_time<<"\t\t\t"<<p[
i].wait_time<<"\t\t"<<endl;
    }
    return time;
}

// Function to print the gannt chart of fcfs
int fcfs_gannt(fcfs f[], int fcfs_count){
```

```
    //Printing the Gantt Chart
    cout<<endl;
    cout<<"Gantt Chart"<<endl;
    cout<<"|";
    for (int i = 0; i < fcfs_count; i++)
    {
        cout<<"P"<<f[i].pid<<"|";
    }
    // cout<<endl;
    // cout<<0;
    // for (int i = 0; i < fcfs_count; i++)
    // {
    //     cout<<"  "<<f[i].completion_time;
    // }
    // cout<<endl;
    return 0;
}

// Function to print the gannt chart of rr2
int rr2_gannt(rr2 r2[], int rr2_count){
    //Printing the Gantt Chart
    // cout<<endl;
    // cout<<"Gantt Chart"<<endl;
    //cout<<"|";
    for (int i = 0; i < rr2_count; i++)
    {
        cout<<"P"<<r2[i].pid<<"|";
    }
    // cout<<endl;
    // cout<<0;
    // for (int i = 0; i < rr2_count; i++)
    // {
    //     cout<<"  "<<r2[i].completion_time;
```

```
    // }
    // cout<<endl;
    return 0;
}

// Function to print the gannt chart of rr4
int rr4_gannt(rr4 r4[], int rr4_count){
    //Printing the Gantt Chart
    // cout<<endl;
    // cout<<"Gantt Chart"<<endl;
    //cout<<"|";
    for (int i = 0; i < rr4_count; i++)
    {
        cout<<"P"<<r4[i].pid<<"|";
    }
    // cout<<endl;
    // cout<<0;
    // for (int i = 0; i < rr4_count; i++)
    // {
    //     cout<<"  "<<r4[i].completion_time;
    // }
    // cout<<endl;
    return 0;
}

// Function to print the gannt chart of sjf
int sjf_gannt(sjf s[], int sjf_count){
    //Printing the Gantt Chart
    //cout<<endl;
    //cout<<"Gantt Chart"<<endl;
    //cout<<"|";
    for (int i = 0; i < sjf_count; i++)
    {
```

```
        cout<<"P"<<s[i].pid<<"|";
    }
    // cout<<endl;
    // cout<<0;
    // for (int i = 0; i < sjf_count; i++)
    // {
    //     cout<<"  "<<s[i].completion_time;
    // }
    //cout<<endl;
    return 0;
}

// Function to print the gannt chart of priority
int priority_gannt(priority p[], int priority_count){
    //Printing the Gantt Chart
    // cout<<endl;
    // cout<<"Gantt Chart"<<endl;
    //cout<<"|";
    for (int i = 0; i < priority_count; i++)
    {
        cout<<"P"<<p[i].pid<<"|";
    }
    // cout<<endl;
    // cout<<0;
    // for (int i = 0; i < priority_count; i++)
    // {
    //     cout<<"  "<<p[i].completion_time;
    // }
    // cout<<endl;
    return 0;
}

// Function to print time in ganntchart
```

```
int time_gannt(fcfs f[], rr2 r2[], rr4 r4[], sjf s[],
priority p[], int fcfs_count, int rr2_count, int
rr4_count, int sjf_count, int priority_count){
    cout<<endl;
    cout<<0;
    for (int i = 0; i < fcfs_count; i++)
    {
        cout<<"  "<<f[i].completion_time;
    }
    for (int i = 0; i < rr2_count; i++)
    {
        cout<<"  "<<r2[i].completion_time;
    }
    for (int i = 0; i < rr4_count; i++)
    {
        cout<<"  "<<r4[i].completion_time;
    }
    for (int i = 0; i < sjf_count; i++)
    {
        cout<<"  "<<s[i].completion_time;
    }
    for (int i = 0; i < priority_count; i++)
    {
        cout<<"  "<<p[i].completion_time;
    }
    cout<<endl;
    return 0;
}

// Function for table
int table(fcfs f[], rr2 r2[], rr4 r4[], sjf s[],
priority p[], int fcfs_count, int rr2_count, int
rr4_count, int sjf_count, int priority_count, int n){
```

```
    int time=0;
    cout<<endl;
    cout<<"Table"<<endl;
    cout<<"PID\tBurst Time\tCompletion Time\tTurn Around
Time\tWait Time\t"<<endl;
    time=dofcfs(f,fcfs_count,time);
    time=dorr2(r2,rr2_count,time);
    time=dorr4(r4,rr4_count,time);
    time=dosjf(s,sjf_count,time);
    time=dopriority(p,priority_count,time);
    // for (int i = 0; i < n; i++)
    // {
    //
cout<<f[i].pid<<"\t"<<f[i].burst_time<<"\t\t"<<f[i].comp
letion_time<<"\t\t"<<f[i].turn_around_time<<"\t\t\t"<<f[
i].wait_time<<"\t\t"<<f[i].response_time<<endl;
    // }

    fcfs_gannt(f,fcfs_count);
    rr2_gannt(r2,rr2_count);
    rr4_gannt(r4,rr4_count);
    sjf_gannt(s,sjf_count);
    priority_gannt(p,priority_count);

time_gannt(f,r2,r4,s,p,fcfs_count,rr2_count,rr4_count,sj
f_count,priority_count);
    return 0;
}




int main()
```

```
{
    int n;
    cout<<"Enter the number of processes: ";
    cin>>n;
    Process p[n];
    for(int i=0;i<n;i++)
    {
        cout<<endl;
        cout<<"Enter the process ID: ";
        cin>>p[i].pid;
        cout<<"Enter the type of process: ";
        cin>>p[i].type;
        switch (p[i].type)
        {
        case 1:
            cout<<"Enter the burst time: ";
            cin>>p[i].burst_time;
            break;
        case 2:
            cout<<"Enter the burst time: ";
            cin>>p[i].burst_time;
            break;
        case 3:
            cout<<"Enter the burst time: ";
            cin>>p[i].burst_time;
            break;
        case 4:
            cout<<"Enter the burst time: ";
            cin>>p[i].burst_time;
            break;
        case 5:
            cout<<"Enter the burst time: ";
            cin>>p[i].burst_time;
```

```
            cout<<"Enter the priority: ";
            cin>>p[i].priority;
            break;
        default:
            break;
        }
    }

    // Counting the number of processes of each type
    int fcfs_count=0;
    int rr2_count=0;
    int rr4_count=0;
    int sjf_count=0;
    int priority_count=0;
    for (int i = 0; i < n; i++)
    {
        if (p[i].type==1)
        {
            fcfs_count++;
        }
        else if (p[i].type==2)
        {
            rr2_count++;
        }
        else if (p[i].type==3)
        {
            rr4_count++;
        }
        else if (p[i].type==4)
        {
            sjf_count++;
        }
        else if (p[i].type==5)
```

```
        {
            priority_count++;
        }
    }

    // Creating arrays for each type of process
    fcfs f[fcfs_count];
    rr2 r2[rr2_count];
    rr4 r4[rr4_count];
    sjf s[sjf_count];
    priority pr[priority_count];

    int n1=0, n2=0, n3=0, n4=0, n5=0;
    // Copying the processes of each type to their
respective arrays
    for (int i = 0; i < n; i++)
    {
        if(p[i].type==1){
            f[n1].pid=p[i].pid;
            f[n1].burst_time= p[i].burst_time;
            n1++;
            // f[i].completion_time=0;
            // f[i].turn_around_time=0;
            // f[i].wait_time=0;
            // f[i].remaining_time=f[i].burst_time;
        }
        else if (p[i].type==2)
        {
            r2[n2].pid=p[i].pid;
            r2[n2].burst_time=p[i].burst_time;
            n2++;
            // r2[i].completion_time=0;
            // r2[i].turn_around_time=0;
```

```
            // r2[i].wait_time=0;
            // r2[i].remaining_time=r2[i].burst_time;
        }
        else if (p[i].type==3)
        {
            r4[n3].pid=p[i].pid;
            r4[n3].burst_time=p[i].burst_time;
            n3++;
            // r4[i].completion_time=0;
            // r4[i].turn_around_time=0;
            // r4[i].wait_time=0;
            // r4[i].remaining_time=r4[i].burst_time;
        }
        else if (p[i].type==4)
        {
            s[n4].pid=p[i].pid;
            s[n4].burst_time=p[i].burst_time;
            n4++;
            // s[i].completion_time=0;
            // s[i].turn_around_time=0;
            // s[i].wait_time=0;
            // s[i].remaining_time=s[i].burst_time;
        }
        else if (p[i].type==5)
        {
            pr[n5].pid=p[i].pid;
            pr[n5].burst_time=p[i].burst_time;
            pr[n5].priority=p[i].priority;
            n5++;
            // pr[i].completion_time=0;
            // pr[i].turn_around_time=0;
            // pr[i].wait_time=0;
            // pr[i].remaining_time=pr[i].burst_time;
```

```
        }
    }


table(f,r2,r4,s,pr,fcfs_count,rr2_count,rr4_count,sjf_co
unt,priority_count,n);

    return 0;
}
```

OUTPUT

```
Enter the number of processes: 8

Enter the process ID: 1
Enter the type of process: 1
Enter the burst time: 6

Enter the process ID: 2
Enter the type of process: 4
Enter the burst time: 10

Enter the process ID: 3
Enter the type of process: 5
Enter the burst time: 4
Enter the priority: 2

Enter the process ID: 4
Enter the type of process: 2
Enter the burst time: 6

Enter the process ID: 5
Enter the type of process: 3
Enter the burst time: 5

Enter the process ID: 6
Enter the type of process: 1
Enter the burst time: 8

Enter the process ID: 7
Enter the type of process: 5
Enter the burst time: 7
Enter the priority: 7

Enter the process ID: 8
Enter the type of process: 4
Enter the burst time: 10
```

```
Table
PID     Burst Time      Completion Time Turn Around Time        Wait Time
1       6               6               6                       0
6       8               14              14                      6
4       6               20              20                      14
5       5               25              25                      20
2       10              35              35                      25
8       10              45              45                      35
3       4               49              49                      45
7       7               56              56                      49

Gantt Chart
|P1|P6|P4|P5|P2|P8|P3|P7|
0  6  14  20  25  35  45  49  56
PS E:\SRM\OS\OS LAB>
```