



**L**OVELY  
**P**ROFESSIONAL  
**U**NIVERSITY

## **School of computer Science**

Progress Report of Simulation based assignment on  
[a program for multilevel queue scheduling algorithm](#)

**Submitted to:** - Lovely Professional University Faculty

name: - Cherry Khosla

**Submitted by:** -

Name: - Akhila Vasireddy

Reg No: - 12114023

Roll No: - RK21SBB45

Section: - K21SB

Group: - 2

Chapter	Content
1.	INTRODUCTION
2.	Problem Definition
3.	Methodology adopted to solve the problem
4.	Algorithm/Steps
5.	Implementation
6.	Results
7.	Conclusion

## INTRODUCTION:-

Multi-level queue scheduling is a type of process scheduling algorithm in which processes are divided into multiple priority queues, and each queue is processed based on a specific

scheduling algorithm. In this approach, processes with a higher priority are given precedence over lower-priority processes.

Each queue may have its own scheduling algorithm, such as round-robin or first-come, first-served (FCFS). Processes are assigned to a queue based on criteria such as their priority level or their resource requirements. The scheduling algorithm for each queue can be different, and the priority level of a process can change dynamically during its execution.

Multi-level queue scheduling is often used in environments where different types of applications or users have different levels of priority. For example, a multi-level queue scheduling algorithm may be used in an operating system to prioritize system-level processes over user-level processes.

This approach can help to improve system performance, as higher-priority processes can be executed more quickly, which can reduce response time and improve overall system efficiency. However, it can also lead to starvation, as lower-priority processes may never get a chance to execute if higher-priority processes are continually being added to the queue.

## **Problem Definition: -**

The problem addressed by multi-level queue scheduling is how to efficiently schedule processes in a system with multiple priority levels. In a multi-level queue scheduling system, processes are divided into different queues based on their priority. Each queue has its own scheduling algorithm, and each queue can have a different priority level.

The goal of multi-level queue scheduling is to ensure that high-priority processes are executed with minimal delay, while also allowing lower-priority processes to be executed when the system is not busy with higher-priority tasks. This helps to ensure that the system remains responsive and efficient, even when multiple processes are running simultaneously.

The challenge of multi-level queue scheduling is to strike a balance between fairness and efficiency. On the one hand, it is important to ensure that all processes are given a chance to execute, regardless of their priority. On the other hand, it is also important to prioritize high-priority tasks to ensure that critical tasks are completed in a timely manner.

## Methodology adopted to solve the problem:-

1. Identifying the different levels of the queue: The first step is to identify the different levels of the queue based on priority, process type, or other criteria.
2. Assigning processes to appropriate queues: Once the levels have been identified, the next step is to assign processes to the appropriate queue based on their priority or type. For example, CPU-bound processes may be assigned to a high-priority queue, while I/O-bound processes may be assigned to a lower-priority queue.
3. Defining scheduling policies: For each queue, a scheduling policy needs to be defined that determines how processes are selected for execution. This may involve using different algorithms such as Round Robin, First-Come-First-Serve, or Shortest Job First.
4. Implementing the scheduler: The next step is to implement the scheduler using the defined scheduling policies. This may involve programming the scheduler in a high-level language such as C or Java.

5. Testing and refinement: Once the scheduler has been implemented, it needs to be tested to ensure that it is working correctly. If any issues are identified, they need to be addressed and the scheduler refined.
6. Evaluation: Finally, the performance of the scheduler needs to be evaluated using appropriate metrics such as response time, throughput, and turnaround time. This evaluation can help identify areas for improvement and guide further refinement of the scheduler.

## **ALGORITHM:-**

Generalized algorithm for multi-level queue scheduling that incorporates round-robin, priority scheduling, and d First Come First Serve (FCFS) scheduling:

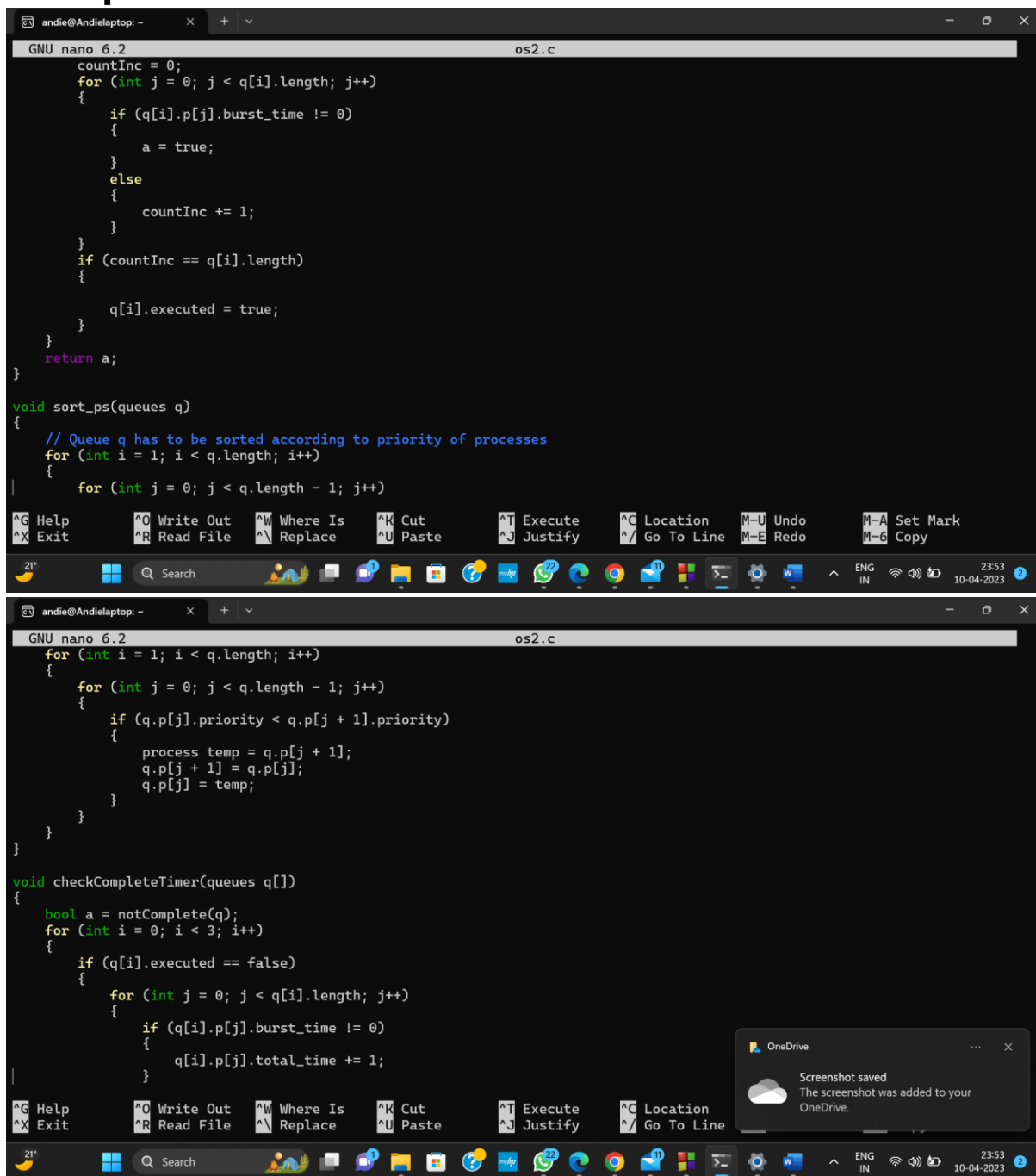
1. Initialize the queues with their respective properties such as time quantum for round-robin and priority level for priority scheduling.
2. Upon arrival of a process, add it to the appropriate queue based on its priority or arrival time.
3. For the highest priority queue, implement priority scheduling until it is empty.
4. For the next highest priority queue, implement round-robin scheduling until it is empty.

5. For the lowest priority queue, implement d FCFS scheduling until it is empty.
6. Repeat steps 3-5 until all queues are empty and all processes have completed execution.

The detailed steps for each type of scheduling algorithm are as follows:

1. Round-Robin Scheduling: a. Set a time quantum for each process to run. b. Execute each process for the specified time quantum, then move it to the next queue. c. If a process completes its execution during the time quantum, remove it from the queue. d. If a process exhausts its time quantum, move it to the next queue.
2. Priority Scheduling: a. Execute the process with the highest priority level until it completes its execution or is preempted by a higher priority process. b. If a higher priority process arrives during the execution of a lower priority process, preempt the lower priority process and execute the higher priority process. c. If a process completes its execution, remove it from the queue.
3. d FCFS Scheduling: a. Execute each process in the order of their arrival time. b. If a process completes its execution, remove it from the queue

## 5. Implementation: -



```
GNU nano 6.2 os2.c
countInc = 0;
for (int j = 0; j < q[i].length; j++)
{
    if (q[i].p[j].burst_time != 0)
    {
        a = true;
    }
    else
    {
        countInc += 1;
    }
}
if (countInc == q[i].length)
{
    q[i].executed = true;
}
return a;
}

void sort_ps(queues q)
{
    // Queue q has to be sorted according to priority of processes
    for (int i = 1; i < q.length; i++)
    {
        for (int j = 0; j < q.length - 1; j++)
        {
            if (q.p[j].priority < q.p[j + 1].priority)
            {
                process temp = q.p[j + 1];
                q.p[j + 1] = q.p[j];
                q.p[j] = temp;
            }
        }
    }
}

void checkCompleteTimer(queues q[])
{
    bool a = notComplete(q);
    for (int i = 0; i < 3; i++)
    {
        if (q[i].executed == false)
        {
            for (int j = 0; j < q[i].length; j++)
            {
                if (q[i].p[j].burst_time != 0)
                {
                    q[i].p[j].total_time += 1;
                }
            }
        }
    }
}
```

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.



```
andie@Andielaptop: ~  
GNU nano 6.2 os2.c  
    }  
    q[i].total_time += 1;  
    }  
}  
}  
  
int main()  
{  
  
    // Initializing 3 queues  
    queues q[3];  
    q[0].priority_start = 7;  
    q[0].priority_end = 9;  
    q[1].priority_start = 4;  
    q[1].priority_end = 6;  
    q[2].priority_start = 1;  
    q[2].priority_end = 3;  
  
    int no_of_processes, priority_of_process, burst_time_of_process;  
    // Prompt User for entering Processes and assigning it to respective queues.  
    cout << "Enter the number of processes\n";  
    cin >> no_of_processes;  
    process p[no_of_processes];  
  
    for (int i = 0; i < no_of_processes; i++)  
    {  
        cout << "Enter the priority of the process\n";  
  
        cout << "Enter the priority of the process\n";  
        cin >> priority_of_process;  
        cout << "Enter the burst time of the process\n";  
        cin >> burst_time_of_process;  
        p[i].priority = priority_of_process;  
        p[i].burst_time = burst_time_of_process;  
        p[i].tt_time = burst_time_of_process;  
        for (int j = 0; j < 3; j++)  
        {  
            if (q[j].priority_start <= priority_of_process && priority_of_process <= q[j].priority_end)  
            {  
                q[j].length++;  
            }  
        }  
  
        for (int i = 0; i < 3; i++)  
        {  
            int len = q[i].length;  
            q[i].p = new process[len];  
        }  
  
        int a = 0;  
        int b = 0;  
        int c = 0;  
  
        for (int i = 0; i < 3; i++)
```

```
andie@Andielaptop: ~  
GNU nano 6.2 os2.c  
{  
    for (int j = 0; j < no_of_processes; j++)  
    {  
        if ((q[i].priority_start <= p1[j].priority) && (p1[j].priority <= q[i].priority_end))  
        {  
            if (i == 0)  
            {  
                q[i].p[a++] = p1[j];  
            }  
            else if (i == 1)  
            {  
                q[i].p[b++] = p1[j];  
            }  
            else  
            {  
                q[i].p[c++] = p1[j];  
            }  
        }  
    }  
    a--;  
    b--;  
    c--;  
    for (int i = 0; i < 3; i++)  
    {  
        cout << "Queue " << i + 1 << " : \t";  
    }  
}
```

21°  
Search  
ENG IN  
23:54  
10-04-2023

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.

```
andie@Andielaptop: ~  
GNU nano 6.2 os2.c  
// While RR on multiple queues is not complete, keep on repeating  
int timer = 0;  
int l = -1;  
int rr_timer = 4;  
int counter = 0;  
int counterps = 0;  
int counterfcfs = 0;  
while (notComplete(q))  
{  
    if (timer == 10)  
    {  
        timer = 0;  
    }  
    l += 1;  
    if (l >= 3)  
    {  
        l = l % 3;  
    }  
    // Process lth queue if its already not executed  
    // If its executed change the value of l  
    if (q[l].executed == true)  
    {  
        cout << "Queue " << l + 1 << " completed\n";  
        l += 1;  
        if (l >= 3)  
        {  
            l = l % 3;  
        }  
    }  
}
```

21°  
Search  
ENG IN  
23:54  
10-04-2023

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.

```
andie@Andielaptop: ~  
GNU nano 6.2 os2.c  
if (l == 0)  
{  
    cout << "Queue " << l + 1 << " in hand\n";  
    // Round Robin Algorithm for q=4  
    if (rr_timer == 0)  
    {  
        rr_timer = 4;  
    }  
  
    for (int i = 0; i < q[l].length; i++)  
    {  
        if (q[l].p[i].burst_time == 0)  
        {  
            counter++;  
            continue;  
        }  
        if (counter == q[l].length)  
        {  
            break;  
        }  
        while (rr_timer > 0 && q[l].p[i].burst_time != 0 && timer != 10)  
        {  
            cout << "Executing queue 1 and " << i + 1 << " process for a unit time. Process has priority of " << q[l].p[i].priority << "\n";  
            q[l].p[i].burst_time--;  
            checkCompleteTimer(q);  
            rr_timer--;  
            timer++;  
        }  
    }  
}  
  
if (timer == 10)  
{  
    break;  
}  
if (q[l].p[i].burst_time == 0 && rr_timer == 0)  
{  
    rr_timer = 4;  
    if (i == (q[i].length - 1))  
    {  
        i = -1;  
    }  
    continue;  
}  
if (q[l].p[i].burst_time == 0 && rr_timer > 0)  
{  
    if (i == (q[i].length - 1))  
    {  
        i = -1;  
    }  
    continue;  
}  
if (rr_timer <= 0)  
{  
    rr_timer = 4;  
    if (i == (q[i].length - 1))  
    {  
        i = -1;  
    }  
}
```

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.

```
GNU nano 6.2 os2.c
}
}
else if (l == 1)
{
    cout << "Queue " << l + 1 << " in hand\n";
    sort_ps(q[l]);
    // Priority Scheduling
    for (int i = 0; i < q[l].length; i++)
    {
        if (q[l].p[i].burst_time == 0)
        {
            counterps++;
            continue;
        }
        if (counterps == q[l].length)
        {
            break;
        }
        while (q[l].p[i].burst_time != 0 && timer != 10)
        {
            cout << "Executing queue 2 and " << i + 1 << " process for a unit time. Process has priority of " << q[l].p[i].priority << "\n";
            q[l].p[i].burst_time--;
            checkCompleteTimer(q);
            timer++;
        }
    }
}
```

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.

21° Search 23:54 10-04-2023

```
GNU nano 6.2 os2.c
if (timer == 10)
{
    break;
}
if (q[l].p[i].burst_time == 0)
{
    continue;
}
}
else
{
    cout << "Queue " << l + 1 << " in hand\n";
    // FCFS
    for (int i = 0; i < q[l].length; i++)
    {
        if (q[l].p[i].burst_time == 0)
        {
            counterfcfs++;
            continue;
        }
        if (counterfcfs == q[l].length)
        {
            break;
        }
        while (q[l].p[i].burst_time != 0 && timer != 10)
        {
            cout << "Executing queue 2 and " << i + 1 << " process for a unit time. Process has priority of " << q[l].p[i].priority << "\n";
            q[l].p[i].burst_time--;
            checkCompleteTimer(q);
            timer++;
        }
    }
}
```

OneDrive  
Screenshot saved  
The screenshot was added to your OneDrive.

21° Search 23:54 10-04-2023



```
andie@Andielaptop: ~  
GNU nano 6.2 os2.c  
#include <iostream>  
using namespace std;  
  
struct process  
{  
    int priority;  
    int burst_time;  
    int tt_time;  
    int total_time = 0;  
};  
  
struct queues  
{  
    int priority_start;  
    int priority_end;  
    int total_time = 0;  
    int length = 0;  
    process *p;  
    bool executed = false;  
};  
  
bool notComplete(queues q[])  
{  
    bool a = false;  
    int countInc = 0;  
    for (int i = 0; i < 3; i++)  
    {  
[ Read 359 lines ]  
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute  
^X Exit      ^R Read File  ^N Replace   ^U Paste      ^J Justify  
^C Location  ^_ Go To Line ^U Undo      ^M-A Set Mark  
^M-E Redo   ^M-G Copy  
21°  Search  [Taskbar icons]  ENG IN  23:53 10-04-2023
```