**Report on Multilevel Queue Scheduling Implementation**

**1. Introduction**

This report explains the implementation of a **Multilevel Queue Scheduling Algorithm**, which categorizes processes into four different queues based on their priority level. The program calculates the waiting time and turnaround time for each process. The Number of processes, their priority levels and their burst time are auto generated. This analysis includes the advantages and disadvantages of each algorithm and discusses the results of the simulation.I have used C++ to implement this M.Q.S.

**2. Overview of the Scheduling Mechanism**

The program organizes processes into four queues (q0, q1, q2, q3) based on priority levels:

1. q0 (Highest Priority) → Round Robin (RR) with a time quantum of 5 seconds.
2. q1 (Second Priority) → Shortest Job First (SJF)
3. q2 (Third Priority) → Shortest Job First (SJF)
4. q3 (Lowest Priority) → First In, First Out (FIFO)

The execution starts with the highest priority queue (q0) and continues downwards if there are no remaining processes or if the queue exceeds the time quantum of 20 seconds. And also I am assuming that arrival time of the each process as 0.

**3. Explanation of the Code**

1. Process Class Implementation
   * Each process has attributes like burst time, remaining time, process ID, priority level, and completion time.
   * The process Id is auto incremented.
   * The class provides methods to update remaining time, completion time, display process details, and calculate turnaround and waiting times.
2. Queue Management
   * Four global queues (q0, q1, q2, q3) store processes based on priority.
   * The generateProcesses() function randomly assigns burst times and priority levels.
3. Scheduling Algorithms
   * First-Come, First-Served (FCFS) for Q3: Processes execute in order of arrival until completion.
   * Shortest Job First (SJF) for Q1 and Q2:Processes are sorted based on remaining burst time and executed accordingly. I have created a vector to sort processes based on burst time. Then I used that sorted vector to re initialize the queue according to the burst time. After that I have called fifo function. This will first executes process with the lowest burst time and so on.
   * Round Robin (RR) for Q0:Each process gets a fixed time slice (5s) before moving to the back of the queue if not completed.
4. Execution Logic
   * The main() function continuously executes processes in q0 → q1 → q2 → q3 until all queues are empty.
   * The function isQueueOver() ensures a queue is fully executed before proceeding to the next level.

**4. Limitations of the Program**

* Does not account for process arrival times dynamically (assumes all processes arrive at time = 0).
* Lack of aging mechanism - Starvation in SJF is not prevented.

**5. Scheduling Algorithm Comparisons**

1. FIFO

* Pros - Simple and easy to implement.
* Cons - Not ideal for real-time systems.

1. SJF

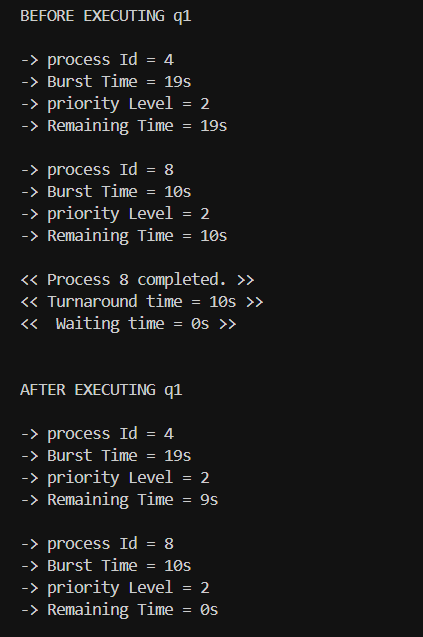
* Pros - Minimizes average waiting time.
* Cons - Requires knowledge of burst times.

1. RR

* Pros - Works well for time sharing systems.
* Cons - Inefficient for processes that could finish quickly.

**6.Explaining the Output**

* I am displaying the details of processes in each queue before and after executing.
* In the below picture I have shown that if a process completes its burst time, then my program shows its turnaround time and the wait time.
* Likewise this program will run until all the processes are completed.
* And also I am not removing the processes after its completion. This way I can show you the accuracy of my implementation using that remaining time of each process even though it leads to some complexity in logic.

****