System Design for the Proposed Solution (Overview of the software architecture, design patterns, data structures, and algorithms used in the project)

The system design for the module that simulates Multilevel Queue (MLQ) CPU Scheduling involve several components, including the software architecture, design patterns, data structures, and algorithms. The following is an overview of each component.

- Software Architecture

The module follow a layered architecture, consisting of presentation layer, application layer, and data layer.

The presentation layer handle the user interface and interaction with the simulation.

The application layer contain the core logic for simulating MLQ CPU Scheduling and managing the queues.

The data layer handle the storage and retrieval of process information.

- Design Patterns

The module utilize the Observer pattern to notify the processes about their turn in the CPU queue.

The Strategy pattern be used to dynamically select the scheduling algorithm based on the priority levels.

- Data Structures

The module use multiple queues to represent the different priority levels in the MLQ.

Each queue be implemented using a data structure such as a linked list or an array.

Each process be represented by a data structure that stores its attributes, such as process ID, priority level, and CPU time required.

- Algorithms

The module employ scheduling algorithms such as Round Robin, First-Come-First-Served, or Shortest Job Next to allocate CPU time to processes within each queue. To prevent CPU starvation, the module implement a mechanism that periodically checks if any lower-priority queues have pending processes and temporarily boosts their priority. The system design also include error handling mechanisms to handle exceptional scenarios, such as invalid input or resource constraints. Additionally, appropriate synchronization mechanisms be implemented to ensure thread safety and prevent conflicts during concurrent execution.

The proposed solution for simulating MLQ CPU Scheduling have a layered software architecture, utilize design patterns like Observer and Strategy, employ data structures such as queues and process representations, and implement scheduling algorithms to allocate CPU time. These components work together to mitigate the issue of CPU starvation while ensuring fairness and optimal performance in the simulation.