Elevator System Design - Coding Interview

Problem Overview

Design and implement an elevator system that can handle passenger requests efficiently. You'll need to create classes that manage elevator operations, handle multiple requests, and ensure thread safety for concurrent operations.

Core Requirements

- Implement an Elevator class that represents a single elevator
- Implement an ElevatorController class that manages elevator operations
- Handle passenger requests (pickup and destination floors)
- Ensure thread-safe operations for concurrent requests
- Implement a basic scheduling algorithm

Difficulty Levels

Easy Level: Single Elevator System

Requirements:

- Design a system with one elevator serving floors 1-10
- The elevator should handle requests to go up or down
- Implement basic states: IDLE, MOVING_UP, MOVING_DOWN, DOOR_OPEN
- Handle pickup requests (floor + direction) and destination requests
- Implement a simple FIFO (First In, First Out) scheduling algorithm

Key Classes to Implement:

class Elevator:
- currentFloor: int
- state: ElevatorState
- targetFloors: List<int>
- methods: moveUp(), moveDown(), openDoor(), closeDoor(), addRequest()

class ElevatorController:
- elevator: Elevator
- methods: requestElevator(floor, direction), processRequests()

Expected Features:

- Basic elevator movement simulation
- Queue management for floor requests
- Simple logging of elevator actions

Medium Level: Multiple Elevator System

Requirements:

- Design a system with 3-5 elevators serving floors 1-20
- Implement an intelligent dispatch algorithm to assign requests to elevators
- Handle concurrent requests from multiple passengers
- Ensure thread safety using appropriate synchronization mechanisms
- Implement elevator optimization (minimize wait time)

Additional Classes:

class ElevatorSystem:
- elevators: List<Elevator>
- requestQueue: Queue<Request>
- methods: assignRequest(), findBestElevator(), balanceLoad()

class Request:
- pickupFloor: int
- destinationFloor: int
- direction: Direction
- timestamp: long

Expected Features:

- Multi-threaded request processing
- Elevator assignment optimization (closest elevator, same direction priority)
- Load balancing between elevators
- Request prioritization
- Comprehensive logging and status reporting

Hard Level: Advanced Enterprise System (Optional Bonus)

Additional Requirements:

- Implement different elevator types (express, local, freight)
- Add maintenance mode and emergency stops
- Implement advanced algorithms (SCAN, LOOK, or custom optimization)
- Add floor restrictions and VIP access
- Performance monitoring and analytics

Technical Specifications

Elevator States

```
enum ElevatorState {
  IDLE,
  MOVING_UP,
  MOVING_DOWN,
  DOOR_OPENING,
  DOOR_OPEN,
  DOOR_CLOSING,
  MAINTENANCE
}
```

Direction Enum

```
enum Direction {
  UP,
  DOWN
}
```

Implementation Guidelines

Thread Safety Considerations

- Use appropriate locks/mutexes for shared resources
- Ensure atomic operations for elevator state changes
- Handle race conditions in request assignment
- Consider using thread-safe collections

Performance Requirements

- System should handle 100+ concurrent requests efficiently
- Response time for elevator assignment should be < 100ms
- Memory usage should remain reasonable under load

Error Handling

- Handle invalid floor requests gracefully
- Implement timeouts for stuck elevators
- Add proper exception handling for concurrent operations