Here's a complete version of the Parking Lot System code, along with a structured explanation of the solution, similar to the one provided for the Transportation System.

---

1. **Problem Statement: Designing a Parking Lot System**

The parking lot system manages parking spots based on vehicle size, ensuring efficient use of limited space. The system should handle vehicle assignments to the nearest available spot, maintain current occupancy status, and manage ticketing for parked vehicles.

---

2. **System Requirements and Solutions**

1. **Parking Spot Management**
    - *Solution*: The `ParkingLot` class manages slots of varying sizes (small, medium, large) and their statuses (available, occupied).
2. **Vehicle Assignment**
    - *Solution*: The system assigns the nearest available parking spot suitable for the vehicle size, ensuring that smaller vehicles can use larger slots but not vice versa.
3. **Current Status Reporting**
    - *Solution*: The `showStatus()` function displays the current status of all parking spots, showing which are occupied or available.
4. **Ticketing System**
    - *Solution*: A ticket is generated when a vehicle is assigned a parking spot, and it can be marked as paid upon exiting.

---

3. **Key Classes, Functions, and Relationships**

1. **Class:** `Slot`
    - Represents a parking slot with attributes for ID, type, and status.
2. **Class:** `Vehicle`
    - Represents vehicles, including their plate number and size.
3. **Class:** `Ticket`
    - Manages parking tickets associated with parked vehicles.
4. **Class:** `ParkingLot`
    - Contains methods to assign and release parking spots and to display the status of all slots.

---

4. **Data Structures Used**

- **Vector** (used in `ParkingLot`):
  - To store and manage multiple parking slots. Allows easy iteration and management of slots.
  - **Complexity**: O(n) for iterating through slots.
- **Unordered Map** (for ticket management):
  - Maps vehicle plate numbers to their corresponding tickets for efficient lookup and management.
  - **Complexity**: O(1) for insertions and lookups.

---

5. **High-Level Design**

- **Slot**: Represents a parking spot with attributes for its type and status.
- **Vehicle**: Represents a vehicle trying to park, including its size.
- **Ticket**: Represents a ticket issued for parked vehicles.
- **ParkingLot**: Central class managing the slots and vehicle assignments.

---

6. **Clarifications to Ask the Interviewer (with Solutions)**

1. **How many vehicle sizes should the system accommodate?**
   - *Solution*: The system currently supports small, medium, and large vehicles, which can be extended if necessary.
2. **Should there be any time limits for parking?**
   - *Solution*: We can implement time-based parking rules that automatically release spots after a certain duration.

---

7. **Open-Ended Questions from the Interviewer**

1. **How would you handle parking for different vehicle types (e.g., electric vehicles)?**
   - *Solution*: Introduce a subclass of `Vehicle` for electric vehicles with additional properties like charging station needs.
2. **How can you improve the system's efficiency?**
   - *Solution*: Implement a priority queue for more sophisticated parking assignment logic during peak times.

---

8. **C++ Code Implementation**

```
#include <iostream>
#include <vector>
#include <unordered_map>
```

```
#include <string>

using namespace std;

enum SlotType { SMALL, MEDIUM, LARGE };
enum SlotStatus { AVAILABLE, OCCUPIED };
enum VehicleSize { CAR, JEEP, TRUCK };
enum TicketStatus { ISSUED, PAID };

class Slot {
public:
    int id;
    SlotType type;
    SlotStatus status;

    Slot(int id, SlotType type) : id(id), type(type), status(AVAILABLE) {}
};

class Vehicle {
public:
    string plateNumber;
    VehicleSize size;

    Vehicle(string plateNumber, VehicleSize size) : plateNumber(plateNumber),
size(size) {}
};

class Ticket {
public:
    int slotId;
    TicketStatus status;

    Ticket(int slotId) : slotId(slotId), status(ISSUED) {}
};

class ParkingLot {
    vector<Slot> slots;
    unordered_map<string, Ticket> tickets;

public:
    ParkingLot(int small, int medium, int large) {
        for (int i = 0; i < small; i++) slots.emplace_back(i, SMALL);
        for (int i = 0; i < medium; i++) slots.emplace_back(i + small, MEDIUM);
        for (int i = 0; i < large; i++) slots.emplace_back(i + small + medium,
LARGE);
    }

    Slot* assignSpot(Vehicle& vehicle) {
        for (SlotType type = vehicle.size; type <= LARGE; type = (SlotType)(type +
1)) {
            for (auto& slot : slots) {
                if (slot.status == AVAILABLE && slot.type == type) {
                    slot.status = OCCUPIED;
                    tickets[vehicle.plateNumber] = Ticket(slot.id);
```

```
                    return &slot;
                }
            }
        }
        return nullptr;
    }

    void releaseSpot(string plateNumber) {
        if (tickets.find(plateNumber) != tickets.end()) {
            int slotId = tickets[plateNumber].slotId;
            tickets.erase(plateNumber);
            for (auto& slot : slots) {
                if (slot.id == slotId) {
                    slot.status = AVAILABLE;
                    break;
                }
            }
        }
    }

    void showStatus() {
        cout << "Slots Status:\n";
        for (const auto& slot : slots) {
            cout << "Slot ID: " << slot.id << ", Type: " << slot.type
                 << ", Status: " << (slot.status == AVAILABLE ? "AVAILABLE" :
"OCCUPIED") << endl;
        }
    }
};

int main() {
    ParkingLot lot(5, 5, 5); // 5 small, 5 medium, 5 large slots

    Vehicle car1("CAR123", CAR);
    Vehicle jeep1("JEEP456", JEEP);
    Vehicle truck1("TRUCK789", TRUCK);

    lot.assignSpot(car1);
    lot.assignSpot(jeep1);
    lot.assignSpot(truck1);

    lot.showStatus(); // Show status of all slots

    lot.releaseSpot(car1.plateNumber); // Release car spot

    cout << "\nAfter releasing a spot:\n";
    lot.showStatus(); // Show updated status of all slots

    return 0;
}
```

Explanation of Code

- `Slot`: Represents parking spots with an ID, type, and status (available or occupied).
- `Vehicle`: Represents vehicles with a plate number and size.
- `Ticket`: Manages the ticket assigned when a vehicle is parked.
- `ParkingLot`: Handles the logic for assigning and releasing parking spots and displays their current status.

This design is optimized for clarity and maintains functionality, making it easy to extend in the future.