Question 3

Conceptual Server Algorithm:

Initialization and Setup

- Create a socket using socket() with IPv4 and TCP.
- Bind the socket to a specific port using bind().
- Listen for incoming connections using listen().
- Print a message indicating the server is listening.

Accepting Client Connections

- Enter a loop to continuously accept client connections using accept().
- Upon accepting a connection, handle the client in a separate function or thread (handle_client() in this case).
- Close the client socket (newSocket) after handling the client.

Handling Clients

- Inside handle_client(newSocket), receive an integer choice from the client indicating their request (1 for catalog display, 2 for book search, etc.).
- Based on the choice, invoke corresponding functions (display_catalog(), search_book(), order_book(), pay_for_book()) to fulfill client requests.
- Each function reads client inputs, processes data (e.g., reads from files, searches, orders, or updates status), and sends appropriate responses back to the client.

Closing Connections

- Close the server socket (sockfd) if needed after exiting the main loop.

Question 3

Application Protocol:

Message Format

- All messages are sent as binary data using read() and write() functions in C.
- Messages are structured as per the requirements of the functions:
    - Integer choices (int) indicating client menu options.
    - Strings (char[]) for titles, statuses, and other textual data.
    - Structs (book structure) for book details.
    - Numeric values (int) for ISBNs, serial numbers, order numbers, etc.

Client Requests

- **Display Catalog**
    - Client sends choice 1.
    - Server responds with the number of books in the catalog (int) and sends each book's details using the book structure.
- **Search for a Book**
    - Client sends choice 2 and either a title (char[]) or an ISBN (int).
    - Server searches for the book and sends back a status (char[]) indicating if the book was found or not.
    - If found, server sends the book details using the book structure.
- **Order a Book**
    - Client sends choice 3 and an ISBN (int) of the book to order.
    - Server checks availability, generates an order number (int), updates the order file, and sends the order number to the client.
- **Pay for a Book**
    - Client sends choice 4 and an order number (int) to pay for.
    - Server updates the payment status in the order file and sends a confirmation status (char[]) back to the client.

Error Handling

- Server sends "not found" status messages (char[]) if books or orders are not found.
- Both client and server handle errors gracefully by printing appropriate messages and exiting if necessary.

Connection Management

- Server maintains persistent connections with clients until the client chooses to exit (choice 5) or an error occurs.