Trivia quiz Game

Overview

The implementation of this Trivia Quiz game uses a concurrent multi-threaded server architecture with blocking I/O operations and multiple clients communicating through TCP sockets, each client has his own thread. The system support simultaneous instances of players and manage 2 type of quizzes, Technology and General knowledge.

Communication Protocol

The app implements a custom message protocol.

1. **Message Format:**
    1. Length-prefixed messages using unit32_t for size
    2. Text-based payload for human readability and easy debugging
    3. Protocol includes commands like “START” & “ENDQUIZ”, and data messages (questions).
2. **Security considerations:**
    1. Buffer size limits to prevent overflow
    2. Message length validation before reading
    3. Socket closure handling
    4. Shared resources mutex-protected

Server Implementation

The server uses a multi-thread concurrent design.

1. **Thread management:**
    1. One thread per client
    2. Thread detachment for autonomous handling
    3. Shared resource protection with shared_mutex
2. **Data structures**
    1. Questions stored in vector, loaded from files, able to scale them however big we want
    2. Player info maintain in vector pair for ease of indexing and get data
    3. Scoreboard implementation with real-time updates

Advantages of implementing the server this way

1. **Concurrent Server:**
    1. *Pros:*
        1. Scales well with multiple clients
        2. Independent client handling
        3. Real-time responsiveness
    2. *Cons:*
        1. Higher resources usage
        2. Complex synchronisation

2. **Text-Based Protocol**
    1. *Pros:*
        1. Easy debugging and logging
        2. Human-readable messages
        3. Simple protocol extension
    2. *Cons:*
        1. Larger messages size
        2. Additional parsing
3. **Shared State Management:**
    1. *Pros:*
        1. Consistent game state across threads
        2. Real-time scoreboard updates
        3. Race condition prevention
    2. *Cons:*
        1. Potential contention on shared resources
        2. Additional synchronisation overhead

Final analysis

1. **Scalability**:
    1. We can declare how many more client we allow, currently we only allow 10
    2. Thread per-client model might not scale well to hundreds of users
2. **Reliability:**
    1. Robust error handling with good logging
    2. Ok handling of client disconnections
    3. Server termination handling
3. **Performance:**
    1. Low latency due to dedicated threads
    2. Minimal blocking operations
    3. Efficient resource sharing with read-write locks
4. **Future improvements:**
    1. DB integration
    2. More game modes
    3. Websocket support for web apps