# Project Scope: Multiplayer Game in C

## 1. Core Technology Stack

- **Language:** C (C99 or C11 standard acceptable, but code style suggests C89/C90 roots).
- **OS Environment:** Linux/Unix (POSIX standard).
- **Libraries:** Standard C Library only (<stdlib.h>, <stdio.h>, <string.h>, <unistd.h>).
- **Networking API:** BSD Sockets (<sys/socket.h>, <netinet/in.h>, <arpa/inet.h>, <netdb.h>).

## 2. Allowed Networking Architectures

The course explicitly covers and permits two main transport layer protocols. The coding agent should choose one based on the game type (e.g., UDP for fast-paced, TCP for turn-based/reliable):

- **TCP (Stream Sockets):**
  - **Usage:** Reliable connection, full-duplex.
  - **Key Functions:** socket(), bind(), listen(), accept(), connect(), send(), recv().
  - **Lifecycle:** Must implement the full handshake and tear-down flow (SYN, ACK, FIN) logic implicitly via these system calls.

- **UDP (Datagram Sockets):**
  - **Usage:** Best-effort, connectionless, lower latency.
  - **Key Functions:** sendto(), recvfrom().

  - **Constraint:** If using UDP, the application layer *must* handle reliability/ordering if the game logic requires it (as UDP does not guaranteed delivery.

## 3. Concurrency & Client Management

The agent must use one of the following three methods to handle multiple players. **Do not mix** them unless necessary.

- **Method A: I/O Multiplexing (Recommended for Single-Threaded Servers)**
  - **Tool:** select().
  - **Mechanism:** Use fd_set, FD_ZERO, FD_SET, and select() with a struct timeval for timeouts. This allows the server to handle multiple sockets in a single loop without blocking.
  - **Constraint:** Do not use epoll (Linux specific) or kqueue (BSD specific); strictly use portable select().
- **Method B: Multi-Threading**
  - **Tool:** POSIX Threads (<pthread.h>).
  - **Mechanism:** One thread per client connection.
  - **Key Functions:** pthread_create(), pthread_join(), pthread_self().
  - **Data Safety:** Arguments passed to threads must be dynamically allocated (using

    malloc) to prevent data races.
- **Method C: Multi-Process (Forking)**
    - **Tool:** fork(.
    - **Mechanism:** Parent process accept()s, then forks a child process to handle the specific client.

## 4. Application Protocol Design (Mandatory)

The agent cannot just send raw strings; it **must** design a custom Application Layer Protocol.

- **Message Format:** Must be explicitly defined.
    - **Header:** Fixed length, containing Message Type (e.g., LOGIN, MOVE, ATTACK) and Payload Length.
    - **Payload:** The actual game data (text or binary).
- **Serialization:**
    - **Text-based:** (e.g., "LOGIN user pass") - easier to debug, used in course examples like POP/HTTP.
    - **Binary Structs:** Sending C structs directly over the wire is permitted but endianness (htons, ntohl) must be handled .
- **State Machine:** The agent should implement a state machine (e.g., New -> Active -> Suspended -> Closed) to manage player states.

## 5. Technical Constraints & "Old School" Requirements

To ensure the code looks like it belongs in the course:

- **IPv4 Priority:** Focus on AF_INET and struct sockaddr_in. IPv6 (struct sockaddr_in6) is mentioned but IPv4 is the primary focus of the code examples.
- **Address Conversion:** Use inet_addr(), inet_ntoa(), or inet_aton() for IP string conversion.
- **Byte Ordering:** STRICTLY use htons()/htonl() for sending integers (like ports or game coordinates) and ntohs()/ntohl() for receiving them.
- **Signal Handling:** Use sigaction or signal to handle server shutdown (e.g., catching SIGINT to close sockets gracefully).

## 6. Out of Scope (Strictly Forbidden)

- **Modern Async I/O:** No io_uring, epoll, or libuv.

- **High-Level Serialization:** No JSON (unless manually parsed), XML, or Protocol Buffers. Use raw C structs or simple delimiters (space/newline).
- **Encryption:** No OpenSSL/TLS implementation (unless the agent writes a raw XOR cipher, standard SSL is out of scope).
- **WebSockets:** No WebSocket handshakes; use raw TCP/UDP.

---

## Implementation Prompt for the Coding Agent

You can use the following prompt to initialize your coding agent:
"Act as a C Network Programming student. I need you to design and implement a Multiplayer Game (Server + Client).
**Constraints:**

1. Use **Standard C libraries** and **BSD Sockets** only.
2. **Architecture:** Client-Server model.
3. **Concurrency:** Use select() for I/O multiplexing OR pthread for multi-threading. Do not use epoll.
4. **Protocol:** Design a custom application protocol (define a C struct for headers and payload).
5. **Data:** Handle Endianness using htons/ntohs.
6. **Address:** Use struct sockaddr_in (IPv4).
7. **Signal Handling:** Gracefully handle SIGINT to close sockets.

**Reference Context:**

- Use gethostbyname or inet_addr for IP resolution.
- If using TCP, ensure full listen()/accept() loop.
- If using UDP, handle sendto()/recvfrom() without connections."