CNT 4504 Computer Networks March 28, 2025

Programming Assignment 1

Toal points: 50

Due Date: April 17th, 2025, 11:59 PM

In this assignment, you will create a two-player Connect 4 game that runs over the network. One machine will act as the server, and another will act as the client. The client moves first. Once a move is made, the server makes the next move, and so forth. The game continues until either a player wins (four pieces in a row) or the board is full (tie).

You are provided with two skeleton C++ files:

- **server_skeleton.cpp** – Contains all the game logic (initializing the board, dropping pieces, checking for wins/ties, printing the board) plus placeholders for the server-side network logic.
- **client_skeleton.cpp** – Contains all the logic needed for the client's user interaction and placeholders for the client-side network logic.

Your task is to implement the socket programming portions so that:

- **The server program:**
  - Creates a listening socket on a specified port.
  - Accepts incoming client connections.
  - After accepting a connection, handles the moves by reading/writing data to the client.
  - Closes the connection when the game ends (win/tie).
  - Waits for the next client connection, repeating indefinitely.
- **The client program:**
  - Connects to the server at the specified hostname/IP and port.
  - Receives the board state and commands from the server.
  - Sends the user's moves (the column in which to drop a piece) back to the server.
  - Displays the result when the game ends and then closes.

Detailed Notes on Each Required Socket Call

Below is a concise list of what functions you should use, why, and their parameters. You will place these calls in the TODO sections of **server_skeleton.cpp** and **client_skeleton.cpp**:

- **`socket(domain, type, protocol)`**
  - **What it does:** Creates an endpoint for network communication.
  - **Typical usage:** `int sock = socket(AF_INET, SOCK_STREAM, 0);`

    - **Why:** We want an IPv4 stream-based (TCP) socket.
    - **Parameters:**
        - `domain` = `AF_INET` (IPv4)
        - `type` = `SOCK_STREAM` (TCP)
        - `protocol` = `0` (auto-select for TCP)
- **`bind(sockfd, (struct sockaddr*)&addr, sizeof(addr))` (server side)**
    - **What it does:** Assigns a local protocol address (IP + port) to the socket.
    - **Why:** So that the operating system knows on which port your server is listening.
    - **Parameters:**
        - `sockfd` is your socket file descriptor.
        - `(struct sockaddr*)&addr` is the pointer to a `sockaddr_in` struct containing `sin_family = AF_INET`, `sin_port = <your port>`, `sin_addr.s_addr = INADDR_ANY` (for "any local address").
        - `sizeof(addr)` is the size of that structure.
- **`listen(sockfd, backlog)` (server side)**
    - **What it does:** Tells the socket to go into listening mode for incoming connection requests.
    - **Why:** In a TCP server, you must first bind to a port, then instruct the OS to queue connection requests.
    - **Parameters:**
        - `sockfd` is your bound socket descriptor.
        - `backlog` is how many pending connections the OS should queue.
- **`accept(sockfd, (struct sockaddr*)&client_addr, &client_len)` (server side)**
    - **What it does:** Blocks until a client attempts to connect, then returns a new socket descriptor for data exchange with that client.
    - **Why:** Accepting a connection is how you retrieve an actual data channel for a single client.
    - **Parameters:**
        - `sockfd` is the "listening" socket.
        - `(struct sockaddr*)&client_addr` is a pointer to a structure that will be filled with the client's address.
        - `&client_len` is the size of that address structure.
- **`connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr))` (client side)**
    - **What it does:** Initiates a connection to a server.
    - **Why:** The client must connect to the server's IP + port.
    - **Parameters:**
        - `sockfd` is your newly created client socket.
        - `(struct sockaddr*)&server_addr` is the server's IP/port.
        - `sizeof(server_addr)` is the size of that structure.
- **`recv(sockfd, buffer, bufsize, flags)`**
    - **What it does:** Reads data from the socket into `buffer`.
    - **Why:** You use `recv()` to get messages (board states, commands, etc.) from the other side.
    - **Parameters:**

        - `sockfd` is the connected socket (from `accept()` on the server side or `connect()` on the client side).
        - `buffer` is where you want to store the incoming bytes.
        - `bufsize` is how many bytes max you can store in `buffer`.
        - `flags` can usually be `0` for a blocking read.
- **`send(sockfd, buffer, bufsize, flags)`**
    - **What it does:** Writes data in `buffer` out over the connected socket.
    - **Why:** You use `send()` to send messages (board updates, commands, etc.) to the other side.
    - **Parameters:**
        - `sockfd` is the connected socket.
        - `buffer` is the data you want to send.
        - `bufsize` is how many bytes you want to send from that buffer.
        - `flags` can usually be `0` for a standard blocking send.
- **`close(sockfd)`**
    - **What it does:** Closes the socket. No further data can be sent or received on this file descriptor.
    - **Why:** When a game finishes or you're shutting down the server, you must close.
    - **Parameters:**
        - `sockfd` is the socket you want to close.

Environment and Execution Details

Important: Your programs must compile and run on one of the school servers listed below. For example, if the server is running on **linprog7.cs.fsu.edu** with port **55000**, then the client should be run with the parameters:

```
128.186.120.191 55000
```

The available school servers are:

- **linprog8.cs.fsu.edu**
  Address: 128.186.120.192
- **linprog5.cs.fsu.edu**
  Address: 128.186.120.189
- **linprog2.cs.fsu.edu**
  Address: 128.186.120.158
- **linprog6.cs.fsu.edu**
  Address: 128.186.120.190
- **linprog4.cs.fsu.edu**
  Address: 128.186.120.181
- **linprog1.cs.fsu.edu**
  Address: 128.186.120.188
- **linprog3.cs.fsu.edu**
  Address: 128.186.120.186

- **linprog7.cs.fsu.edu**
  Address: 128.186.120.191

What to Turn In

1. **Completed server.cpp:**
   - All networking TODO parts filled in.
   - Must compile and run correctly on your system.
   - Must allow multiple consecutive client connections (one game ends, wait for the next connection).
2. **Completed client.cpp:**
   - All networking TODO parts filled in.
   - Must compile and run correctly, connecting to the server, playing a game, and then exiting.
3. **A project report:**
   - That includes a summary of the commands and arguments you used for each executable, any issues and difficulties, and the description of your testing of your program.

Compilation and Testing

Your two C++ files must pass the following compilation commands on the FSU linprog servers:

- Compile the server:
  `g++ -o server Server.cpp`
- Compile the client:
  `g++ -o client Client.cpp`

You must be able to play the game using two terminal windows, one running the server executable and one running the client executable.

Grading Criteria (approximate)

- **(40%)** Correctness of socket calls (creation, binding, listening, acceptance, connecting, etc.).
- **(30%)** Proper reading/writing of data (adhering to the simple text-based protocol).
- **(10%)** Proper error handling (checking return values of system calls).
- **(10%)** Code readability and comments.
- **(10%)** Ability to handle multiple games in sequence on the server side.

Note: A program that cannot compile will receive no more than **15 points**. Make sure your code compiles on one of the school servers before submission.