Simple C Chat Application

This project implements a multi-client chat server and a corresponding client using the C programming language and POSIX sockets. It utilizes **I/O multiplexing** via `select()` to handle multiple concurrent connections and terminal input without the need for multi-threading.

Architecture

The application follows a standard Client-Server Architecture.

- The server acts as a central hub. It listens for new connections, manages a list of active clients, and broadcasts or directs messages based on user input.
- The clients connect to the server, and each authenticates with a username, and allows the user to send and receive messages in real-time.

I/O Multiplexing

Both the server and client use the `select()` system call. This allows the programs to monitor multiple file descriptors (sockets and STDIN) simultaneously, ensuring the application remains responsive to both network data and user keyboard input.

Communication Protocol

Authentication

Upon connection, the client must immediately send an authentication string to the server: `@authenticate <username>`

If the server receives any other string first, the connection is terminated for security and protocol consistency.

Messaging

- Public Broadcast: Any message sent by a client (that does not start with `@`) is prepended with the sender's username and broadcasted to all connected clients.
- Private Whisper: If a message begins with `@username`, the server parses the target name and only forwards the message to that specific client.
- Exit Command: Typing `!exit` in either the client or server terminal will gracefully close the respective application.

Server Implementation Details

Client Management

The server maintains an array of `client_T` structures. When a client disconnects: 1. The file descriptor is closed. 2. The descriptor is marked as `-1`. 3. The server performs a smart array shift to remove the "holes" in the array and keep the `client_count` accurate.

Whisper Logic

The server parses the incoming message. If the first character is `@`, it uses `strtok` to extract the target username and compares it against the active client list using `strcmp`.

Build and Execution

The project includes a `Makefile` for streamlined compilation and debugging.

Compilation

To build the standard versions:

```
make server
make client
```

To build with advanced debugging and memory sanitization enabled:

```
make server-debug
make client-debug
```

Running the Server

The server requires a port number as an argument:

```
./hw3server 8080
```

Running the Client

The client requires the server address, port, and a chosen username:

```
./hw3client 127.0.0.1 8080 Alice
```

Cleanup

To remove the compiled binaries:

```
make clean
```

---

6. Limitations & Notes

- Buffer Management: The application uses fixed-size buffers (`MAX_MESSAGE`).
- Linear Search: The server finds whisper targets using a linear search ($O(n)$). This is efficient for small `MAX_CLIENTS` but would require a hash map for very large-scale applications.
- Reliability: The current implementation assumes a successful `send()`. In a production environment, checking the return value of `send()` to ensure the full buffer was transmitted is recommended.