C++ Chat Application

This is a multi-client chat application built in modern C++ (17). It features a non-blocking server capable of handling multiple simultaneous client connections using `epoll`.

The project is designed with a clean, modular architecture, emphasizing SOLID principles and testability.

Features

- **Concurrent Server**: A single-threaded server using `epoll` to manage multiple clients concurently.
- **Multi-threaded Client**: A responsive command-line client that handles user input and network messages concurrently.
- **Custom Binary Protocol**: A robust and extensible binary protocol for clear and efficient client-server communication.
- **SOLID Design**: Code is decoupled into logical components (`common`, `server`, `client`) and utilizes interfaces for testability.
- **Unit & Integration Testing**: The project is structured to support unit tests with Google Test.

Project Structure

The project is organized into distinct components to enforce a clean separation of concerns.

```
axn_chat_app/
├── CMakeLists.txt
├── README.md
├── client
│   ├── CMakeLists.txt
│   ├── include
│   │   └── client
│   │       ├── chat_client.h
│   │       └── server_connection.h
│   └── src
│       ├── chat_client.cpp
│       ├── main.cpp
│       └── server_connection.cpp
├── common
│   ├── CMakeLists.txt
│   ├── include
│   │   └── common
│   │       ├── logger.h
│   │       ├── protocol.h
│   │       └── socket.h
│   └── src
│       ├── protocol.cpp
│       └── socket.cpp
├── server
│   ├── CMakeLists.txt
│   ├── include
│   │   └── server
```

```
│   │       ├── client_manager.h
│   │       ├── client_session.h
│   │       ├── epoll_manager.h
│   │       └── server.h
│   └── src
│       ├── client_manager.cpp
│       ├── client_session.cpp
│       ├── epoll_manager.cpp
│       ├── main.cpp
│       └── server.cpp
└── tests
    ├── CMakeLists.txt
    ├── client
    │   ├── CMakeLists.txt
    │   └── chat_client_test.cpp
    ├── common
    │   ├── CMakeLists.txt
    │   ├── protocol_test.cpp
    │   └── socket_test.cpp
    └── server
        ├── CMakeLists.txt
        ├── client_manager_test.cpp
        └── server_integration_test.cpp
```

Prerequisites

This project is designed for a Linux environment. You will need the following tools installed:

- A C++17 compliant compiler (e.g., `g++`)
- `CMake` (version 3.14 or newer)
- `make`
- `gdb` (Optional)

You can install these on a Debian-based system (like Ubuntu) with:

```
sudo apt-get update
sudo apt-get install build-essential cmake
sudo apt-get install g++
```

How to Build

Building Natively

Follow these steps to build the server, client, and test executables.

1. **Clone the repository:**

```
git clone https://github.com/datdd/axn_chat_app.git
cd chat_app
```

2. **Create and navigate to the build directory:**

```
mkdir build
cd build
```

3. **Configure the project with CMake:**

```
cmake ..
```

4. **Build all targets:**

```
make
```

Building a Specific Component

If you only want to build a single part of the project (e.g., just the client), you can specify the target.

- **Build only the client:**

```
# From the 'build' directory
make chat_client
```

- **Build only the server:**

```
# From the 'build' directory
make chat_server
```

Build using Docker

1. **Clone the repository:**

```
git clone https://github.com/datdd/axn_chat_app.git
cd chat_app
```

2. **Build Docker Image**

```
cd axn_chat_app/.devcontainer
docker build -t axn_chat_app .
```

3. **Run Docker Container**

```
docker run -it --rm -v /path/on/host:/path/in/container --name my_container
my_image
```

Explanation:

- -v /path/on/host:/path/in/container: Maps a directory from your host system to the container. Replace /path/on/host with your actual folder location and /path/in/container with the desired location inside the container.

Example: Run Docker on Windows

```
docker run -it --rm -v
C:\\Windows\\path\\example\\axn_chat_app:/workspace/axn_chat_app --name
axn_chat_app axn_chat_app
```

4. **Build**

Follow steps 2, 3, 4 in the `Building Natively` section.

How to Run

You must run the server first, then connect one or more clients.

1. **Start the Server:**

From the `build` directory, run:

```
./server/chat_server <port>
```

For example, to run the server on port 8080:

```
./server/chat_server 8080
```

2. **Start a Client:**

Open a **new terminal window** for each client you want to connect. From the `build` directory, run:

```
./client/chat_client <host_ip> <port> <username>
```

- `<host_ip>`: The IP address of the machine running the server. Use `127.0.0.1` if running on the same machine.
- `<port>`: The port the server is listening on (e.g., 8080).
- `<username>`: The username you want to use in the chat.

Example (Client 1):

```
./client/chat_client 127.0.0.1 8080 Client01
```

Example (Client 2, in another terminal):

```
./client/chat_client 127.0.0.1 8080 Client02
```

3. **Chat!**

   Type messages in any client terminal and press Enter. They will be broadcast to all other connected clients. To disconnect a client, type `/exit`.

Run on Docker

We will run the chat_server and multiple chat_client on separated containers. Make sure these containers running on the same Network. If you haven't built a Docker image, please follow the `Build using Docker` section.

1. **Create a Docker Network with a Subnet**

```
docker network create --subnet=192.168.1.0/24 local_network
```

2. **Run a Container with a Static IP**

   - Run `chat_server`

```
docker run -it --rm `
    -v C:\\Windows\\path\\example\\axn_chat_app:/workspace/axn_chat_app
`
    --name chat_server `
    --net local_network --ip 192.168.1.100 `
    axn_chat_app
```

   - Run `chat_client`

- Client 01

```
docker run -it --rm `
    -v C:\\Windows\\path\\example\\axn_chat_app:/workspace/axn_chat_app
`
    --name chat_client_01 `
    --net local_network --ip 192.168.1.201 `
    axn_chat_app
```

- Client 02

```
docker run -it --rm `
    -v C:\\Windows\\path\\example\\axn_chat_app:/workspace/axn_chat_app
`
    --name chat_client_02 `
    --net local_network --ip 192.168.1.202 `
    axn_chat_app
```

3. **Run server and client**

   Navigate to the build directory:

   ```
   cd /workspace/axn_chat_app/build
   ```

   Follow the sections `Start the Server` and `Start the Client` to start chat_server and chat_client.

How to Run Tests

The project is configured to use Google Test. The tests can be run after building the project.

```
# From the 'build' directory
ctest
```

This will execute all unit and integration tests and report the results.

Debugging with DDB

Follow the `How to Build` steps but configure the project in Debug mode:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

Debugging the Server

1. **Launch the server with GDB:**

```
# From the 'build' directory
gdb --args ./chat_server 8080
```

2. **Set breakpoints:**

```
# Set a breakpoint at the function that handles new connections
(gdb) break chat_app::server::Server::handle_new_connection

# Set a breakpoint at the function that processes incoming messages
(gdb) break chat_app::server::Server::process_message
```

3. **Run the program:**

```
(gdb) run
```

4. **Inspect state:**

```
# Print the value of a variable
(gdb) print variable_name
```

Debugging the Client

1. **Launch the client with GDB:**

```
gdb --args ./chat_client 192.168.1.100 8080 username
```

2. **Run the program:**

```
(gdb) run
```