Course Project

Project Topic: **“26. Server”**

Author:
Student of Group CM-2
Ivan Tiulpa

December 1, 2024

Contents

1 Overview **2**

2 Project Directory Structure **2**

3 Project Structure **2**
3.1 File Contents 2
3.2 Used Libraries 3

4 Class Descriptions **4**
4.1 `Server` Class 4
4.2 `ServerSocket` Class 4
4.3 `ServerClient` Class 5
4.4 `CrossPlatformClient` Class 7
4.5 `UnixSocketClient` Class 7
4.6 `SocketClient` Class 7

1. Overview

This project presents a client-server system that supports connections of an arbitrary number of clients. Each client, upon successful connection, can immediately send messages to the server of type `wchar_t`. The server forwards these messages to all other connected clients except the sender. Essentially, this is an implementation of a group chat.

Upon connection, each client is assigned a name. For example, if there are already 3 clients connected to the server, the 4th client will be assigned the name `[Client 4]`.

If a client does not like their assigned name, they can change it using the command:

```
/name new_name
```

After this, their name will change to `[new_name]`.

2. Project Directory Structure

```
server_project/
├── Client_files/
│   ├── client_start.cpp
│   ├── CrossPlatformClient.h
│   ├── CrossPlatformClient.cpp
│   ├── UnixSocketClient.cpp
│   ├── UnixSocketClient.h
│   └── SocketClient.h
├── Server_files/
│   ├── server_start.cpp
│   ├── Server.h
│   ├── Server.cpp
│   ├── ServerClient.cpp
│   ├── ServerClient.h
│   ├── ServerSocket.cpp
│   └── ServerSocket.h
├── .gitignore
├── CMakeLists.txt
├── make.sh
└── server_test.cpp
```

3. Project Structure

3.1. File Contents

- **Client_files/**
  - **client_start.cpp**: The file that includes **CrossPlatformClient.h** and passes it the IP and PORT that the user enters in the console.
  - **CrossPlatformClient.cpp** and **CrossPlatformClient.h**: Define the class `CrossPlatformClient`, which manages socket operations on the client side.
  - **UnixSocketClient.cpp** and **UnixSocketClient.h**: Define the class `UnixSocketClient` for working with Unix-based sockets.
  - **SocketClient.h**: Contains the abstract class from which `UnixSocketClient` inherits.

- **Server_files/**
  - **server_start.cpp**: The file that includes **Server.h** and passes it the IP, PORT, and BACKLOG that the user enters in the console.
  - **Server.cpp**: Contains the implementation of the `Server` class, which manages client connections and message handling.
  - **ServerClient.cpp** and **ServerClient.h**: Define the `ServerClient` class responsible for interacting with the client's socket.
  - **ServerSocket.cpp** and **ServerSocket.h**: Define the `ServerSocket` class responsible for creating, binding, and listening on sockets.
- **.gitignore**: File for excluding certain files and directories from version control.
- **CMakeLists.txt**: Configuration file for building the project using CMake.
- **make.sh**: Script that compiles the server and client and stores them in a newly created `build` directory.
- **server_test.cpp**: File where you can run both the server and `n` clients on your own machine to test the project.

3.2. Used Libraries

Standard Libraries

`#include <iostream>`: For input and output operations.

`#include <string>`: Support for working with strings.

`#include <map>`: For storing and managing client names associated with their sockets.

`#include <vector>`: For managing a dynamic array of file descriptors for polling.

`#include <cstring>`: Functions for manipulating C-strings.

`#include <cerrno>`: Handling error numbers and providing error messages.

`#include <thread>`: Used for handling client requests from the server and running `cin >>` for inputting messages to be sent to the server.

POSIX Libraries

`#include <netdb.h>`: Definitions for network operations.

`#include <unistd.h>`: Access to the POSIX operating system API.

`#include <arpa/inet.h>`: Definitions for internet operations.

`#include <csignal>`: Functions for signal handling.

`#include <sys/wait.h>`: Macros related to process termination.

`#include <poll.h>`: The `poll` function for monitoring multiple file descriptors.

4. Class Descriptions

4.1. `Server` Class

Manages client connections and message handling. Uses `poll` to monitor multiple file descriptors and processes incoming messages from clients.

Attributes

- `ServerSocket *server`: Pointer to a `ServerSocket` object.
- `ServerClient *client`: Pointer to a `ServerClient` object.
- `std::map<int, std::wstring> clients_names`: A map of client socket file descriptors to client names.
- `std::vector<pollfd> fds`: A vector of file descriptors for polling.
- `int sockfd`: The server's file descriptor.
- `const char *IP`: The server's IP address.
- `const char *PORT`: The server's port number.

Methods

- `Server(const char *IP, const char *PORT, const int backlog)`: Constructor that initializes the server with the given IP, port, and backlog.
- `~Server()`: Destructor that cleans up resources.
- `void addClient(int new_socket)`: Adds a new client to the server.
- `void readMsgFromUser(int index_of_skipped_client)`: Reads a message from a client.
- `void handleClientDisconnection(int index_of_skipped_client, int skipped_client_so` Handles a client's disconnection.
- `void handleClientNameChange(int skipped_client_socket, wchar_t buffer[BUFFER_SIZE` Handles a client's name change.
- `void handleClientMessage(int index_of_skipped_client, int skipped_client_socket, wchar_t buffer[BUFFER_SIZE])`: Handles messages from clients.
- `void handle_connections()`: Manages connections and processes incoming messages.

4.2. `ServerSocket` Class

Responsible for creating the socket, binding it, and listening. Initializes the socket, binds it to an address, and starts listening for incoming connections.

Attributes

- `int sockfd`: The socket file descriptor.
- `const int backlog`: The maximum length of the queue of pending connections.
- `const char *IP`: The server's IP address.
- `const char *PORT`: The server's port number.

Methods

- `ServerSocket(const char* IP, const char* PORT, const int backlog)`: Constructor that initializes the server socket with the given parameters.
- `~ServerSocket()`: Destructor that cleans up resources.
- `int runSocket()`: Runs the socket and returns its file descriptor.
- `static void print_connection(sockaddr_storage addr, std::string &message)`: Outputs connection information.
- `addrinfo init_hints()`: Initializes the `addrinfo` structure with hints.
- `addrinfo* init_getaddrinfo(const addrinfo &hints)`: Performs `getaddrinfo` with the given hints.
- `int bind_socket_to_addr(addrinfo *result_of_getaddr)`: Binds the socket to an address.
- `void bind_socket(addrinfo *result_of_getaddr)`: Binds the socket.
- `void start_listen()`: Starts listening on the socket.
- `void print_bound_adress(std::string message) const`: Outputs the bound address.
- `void kill_needless_processes()`: Terminates unnecessary processes.
- `static void sigchld_handler(int s)`: Signal handler for `SIGCHLD`.

4.3. `ServerClient` Class

Responsible for client-side operations and communication via sockets.

Attributes

- `int sockfd`: The socket file descriptor.

Methods

- `ServerClient(int sockfd)`: Constructor that initializes the client with the given file descriptor.
- `static void sendToClientsExceptOne(std::vector<pollfd> fds, wchar_t buffer[BUFFER int index_of_excepted_user)`: Sends a message to all clients except one.
- `static void sendMsgToClient(int client_sockfd, wchar_t buffer[BUFFER_SIZE])`: Sends a message to a specific client.
- `int acceptClient() const`: Accepts a new client connection.
- `static int receiveFromClient(int client_sockfd, wchar_t buffer[BUFFER_SIZE])`: Receives a message from a client.
- `static void handleError(const std::string& message, bool condition)`: Handles errors.

4.4. `CrossPlatformClient` Class

Manages client-side operations and communication via sockets. Connects to the server, sends messages, and receives responses.

Attributes

- `SocketClient *client`: Pointer to a `SocketClient` object.
- `int sockfd`: The socket file descriptor.
- `const char *IP`: The server's IP address.
- `const char *PORT`: The server's port number.

Methods

- `CrossPlatformClient(SocketClient *c, const char *IP, const char *PORT)`: Constructor that initializes the client with the given socket client, IP, and port.
- `void connectToServer()`: Connects to the server.
- `void initSockfd()`: Initializes the socket file descriptor.
- `void sendToServer(std::wstring &message)`: Sends a message to the server.
- `void receiveFromServer()`: Receives messages from the server.
- `void sendHandler()`: Handles sending messages.
- `void receiveHandler()`: Handles receiving messages.
- `void closeConnection()`: Closes the connection.
- `void start()`: Starts the client.
- `~CrossPlatformClient()`: Destructor that cleans up resources.

4.5. `UnixSocketClient` Class

Provides Unix-based socket operations for the client.

Methods

- `void connectToServer(const char *IP, const char *PORT)`: Connects to the server.
- `int getSocket()`: Returns the socket file descriptor.
- `int sendMessage(const std::wstring &message)`: Sends a message to the server.
- `int receiveMessage(wchar_t buffer[BUFFER_SIZE])`: Receives a message from the server.

4.6. `SocketClient` Class

Abstract base class for socket client operations.

Methods

- `virtual void connectToServer(const char *IP, const char *PORT) = 0`: Pure virtual method for connecting to the server.
- `virtual int getSocket() = 0`: Pure virtual method for obtaining the socket file descriptor.
- `virtual int sendMessage(const std::wstring &message) = 0`: Pure virtual method for sending a message to the server.
- `virtual int receiveMessage(wchar_t buffer[BUFFER_SIZE]) = 0`: Pure virtual method for receiving a message from the server.