Курсовий Проєкт

Тема проєкту: **«26. Server»**

Виконав:
студент групи КМ-2
Тюльпа Іван Юрійович

1 грудня 2024 р.

Зміст

1 Огляд **2**

2 Структура директорії проєкту **2**

3 Структура проєкту **2**
3.1 Вміст файлів 2
3.2 Використані бібліотеки 3

4 Опис класів **4**
4.1 Клас `Server` 4
4.2 Клас `ServerSocket` 4
4.3 Клас `ServerClient` 5
4.4 Клас `CrossPlatformClient` 7
4.5 Клас `UnixSocketClient` 7
4.6 Клас `SocketClient` 8

1. Огляд

Цей проєкт представляє клієнт-серверну систему, що підтримує підключення довільної кількості клієнтів. Кожен клієнт після вдалого підключення може одразу надсилати повідомлення на сервер типу `wchar_t`. Сервер пересилає ці повідомлення всім іншим підключеним клієнтам, окрім відправника. По суті, це реалізація групового чату.

Після підключення кожному клієнту надається ім'я. Наприклад, якщо на сервер вже підключилися 3 клієнти, то 4-му клієнту після підключення буде надано ім'я `[Client 4]`.

Якщо клієнту не подобається його ім'я, він може змінити його командою:

```
/name new_name
```

Після цього його ім'я зміниться на `[new_name]`.

2. Структура директорії проєкту

```
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. Структура проєкту

3.1. Вміст файлів

- **Client_files/**
  - **client_start.cpp**: файл, який підключає **CrossPlatformClient.h**, та передає йому IP, PORT, які користувач вводить в консолі.
  - **CrossPlatformClient.cpp** та **ServerClient.h**: визначає клас `CrossPlatformClient`, який керує операціями з сокетами на стороні клієнта.
  - **UnixSocketClient.cpp** та **UnixSocketClient.h**: визначає клас `UnixSocketClient` для роботи з сокетами на основі Unix.

  - **SocketClient.h**: містить абстрактний клас, наслідком якого є клас `UnixSocketClient`.

- **Server_files/**
  - **server_start.cpp**: файл, який підключає **Server.h**, та передає йому IP, PORT BACKLOG, які користувач вводить в консолі.
  - **Server.cpp**: містить реалізацію класу `Server`, який керує клієнтськими з'єднаннями та обробкою повідомлень.
  - **ServerClient.cpp** та **ServerClient.h**: визначають клас `ServerClient`, що відповідає за взаємодію з клієнтським сокетом.
  - **ServerSocket.cpp** та **ServerSocket.h**: визначають клас `ServerSocket`, який відповідає за створення, прив'язку та прослуховування сокетів.
- **.gitignore**: файл для виключення певних файлів та директорій з контролю версій.
- **CMakeLists.txt**: файл конфігурації для збірки проєкту за допомогою CMake.
- **make.sh**: файл, що компілює сервер та клієнт, та зберігає їх у новоствореній директорії build.
- **server_test.cpp**: файл, в якому можна запустити одночасно сервер і n клієнтів на власній машині, щоб протестувати сам проєкт.

3.2. Використані бібліотеки

Стандартні бібліотеки

`#include <iostream>`: для операцій вводу та виводу.

`#include <string>`: підтримка для роботи зі строками.

`#include <map>`: для зберігання та керування іменами клієнтів, асоційованими з їхніми сокетами.

`#include <vector>`: для керування динамічним масивом файлових дескрипторів для опитування.

`#include <cstring>`: функції для маніпуляцій з C-рядками.

`#include <cerrno>`: обробка номерів помилок та надання повідомлень про помилки.

`#include <thread>`: використовується для обробки клієнтом запитів від сервера, та запуска cin », для вводу повідомлень, що будуть відправлені на сервер.

POSIX бібліотеки

`#include <netdb.h>`: визначення для операцій з мережею.

`#include <unistd.h>`: доступ до API операційної системи POSIX.

`#include <arpa/inet.h>`: визначення для інтернет-операцій.

`#include <csignal>`: функції для обробки сигналів.

`#include <sys/wait.h>`: макроси, пов'язані з завершенням процесів.

`#include <poll.h>`: функція `poll` для моніторингу кількох файлових дескрипторів.

4. Опис класів

4.1. Клас `Server`

Керує з'єднаннями клієнтів та обробкою повідомлень. Використовує `poll` для моніторингу кількох файлових дескрипторів та обробляє вхідні повідомлення від клієнтів.

Атрибути

- `ServerSocket *server`: вказівник на об'єкт `ServerSocket`.
- `ServerClient *client`: вказівник на об'єкт `ServerClient`.
- `std::map<int, std::wstring> clients_names`: відображення файлових дескрипторів сокетів клієнтів на імена клієнтів.
- `std::vector<pollfd> fds`: вектор файлових дескрипторів для опитування.
- `int sockfd`: файловий дескриптор сервера.
- `const char *IP`: IP-адреса сервера.
- `const char *PORT`: номер порту сервера.

Методи

- `Server(const char *IP, const char *PORT, const int backlog)`: конструктор, який ініціалізує сервер із заданими IP, портом та чергою підключень.
- `~Server()`: деструктор, що очищує ресурси.
- `void addClient(int new_socket)`: додає нового клієнта до сервера.
- `void readMsgFromUser(int index_of_skipped_client)`: зчитує повідомлення від клієнта.
- `void handleClientDisconnection(int index_of_skipped_client, int skipped_client_so` обробляє відключення клієнта.
- `void handleClientNameChange(int skipped_client_socket, wchar_t buffer[BUFFER_SIZE` обробляє зміну імені клієнта.
- `void handleClientMessage(int index_of_skipped_client, int skipped_client_socket, wchar_t buffer[BUFFER_SIZE])`: обробляє повідомлення від клієнтів.
- `void handle_connections()`: керує з'єднаннями та обробляє вхідні повідомлення.

4.2. Клас `ServerSocket`

Відповідає за створення сокета, його прив'язку та прослуховування. Ініціалізує сокет, прив'язує його до адреси та починає прослуховувати вхідні з'єднання.

Атрибути

- `int sockfd`: файловий дескриптор сокета.
- `const int backlog`: максимальна довжина черги очікуваних з'єднань.
- `const char *IP`: IP-адреса сервера.
- `const char *PORT`: номер порту сервера.

Методи

- `ServerSocket(const char* IP, const char* PORT, const int backlog)`: конструктор, що ініціалізує серверний сокет із заданими параметрами.
- `~ServerSocket()`: деструктор, що очищує ресурси.
- `int runSocket()`: запускає сокет і повертає його файловий дескриптор.
- `static void print_connection(sockaddr_storage addr, std::string &message)`: виводить інформацію про з'єднання.
- `addrinfo init_hints()`: ініціалізує структуру `addrinfo` з підказками.
- `addrinfo* init_getaddrinfo(const addrinfo &hints)`: виконує `getaddrinfo` з заданими підказками.
- `int bind_socket_to_addr(addrinfo *result_of_getaddr)`: прив'язує сокет до адреси.
- `void bind_socket(addrinfo *result_of_getaddr)`: прив'язує сокет.
- `void start_listen()`: починає прослуховування на сокеті.
- `void print_bound_adress(std::string message) const`: виводить прив'язану адресу.
- `void kill_needless_processes()`: завершує непотрібні процеси.
- `static void sigchld_handler(int s)`: обробник сигналу `SIGCHLD`.

4.3. Клас `ServerClient`

Відповідає за операції на стороні клієнта та комунікацію через сокети.

Атрибути

- `int sockfd`: файловий дескриптор сокета.

Методи

- `ServerClient(int sockfd)`: конструктор, що ініціалізує клієнта з заданим файловим дескриптором.
- `static void sendToClientsExceptOne(std::vector<pollfd> fds, wchar_t buffer[BUFFER int index_of_excepted_user)`: надсилає повідомлення всім клієнтам, крім одного.
- `static void sendMsgToClient(int client_sockfd, wchar_t buffer[BUFFER_SIZE])`: надсилає повідомлення конкретному клієнту.
- `int acceptClient() const`: приймає нове з'єднання від клієнта.
- `static int receiveFromClient(int client_sockfd, wchar_t buffer[BUFFER_SIZE])`: отримує повідомлення від клієнта.
- `static void handleError(const std::string& message, bool condition)`: обробляє помилки.

4.4. Клас `CrossPlatformClient`

Керує операціями на стороні клієнта та комунікацією через сокети. Підключається до сервера, надсилає повідомлення та отримує відповіді.

Атрибути

- `SocketClient *client`: вказівник на об'єкт `SocketClient`.
- `int sockfd`: файловий дескриптор сокета.
- `const char *IP`: IP-адреса сервера.
- `const char *PORT`: номер порту сервера.

Методи

- `CrossPlatformClient(SocketClient *c, const char *IP, const char *PORT)`: конструктор, який ініціалізує клієнт із заданим сокетним клієнтом, IP та портом.
- `void connectToServer()`: підключається до сервера.
- `void initSockfd()`: ініціалізує файловий дескриптор сокета.
- `void sendToServer(std::wstring &message)`: надсилає повідомлення на сервер.
- `void receiveFromServer()`: отримує повідомлення від сервера.
- `void sendHandler()`: обробляє відправку повідомлень.
- `void receiveHandler()`: обробляє отримання повідомлень.
- `void closeConnection()`: закриває з'єднання.
- `void start()`: запускає клієнт.
- `~CrossPlatformClient()`: деструктор, що очищує ресурси.

4.5. Клас `UnixSocketClient`

Забезпечує операції з сокетами на основі Unix для клієнта.

Методи

- `void connectToServer(const char *IP, const char *PORT)`: підключається до сервера.
- `int getSocket()`: повертає файловий дескриптор сокета.
- `int sendMessage(const std::wstring &message)`: надсилає повідомлення на сервер.
- `int receiveMessage(wchar_t buffer[BUFFER_SIZE])`: отримує повідомлення від сервера.

4.6. Клас `SocketClient`

Абстрактний базовий клас для операцій сокетного клієнта.

Методи

- `virtual void connectToServer(const char *IP, const char *PORT) = 0`: чисто віртуальний метод для підключення до сервера.
- `virtual int getSocket() = 0`: чисто віртуальний метод для отримання файлового дескриптора сокета.
- `virtual int sendMessage(const std::wstring &message) = 0`: чисто віртуальний метод для надсилання повідомлення на сервер.
- `virtual int receiveMessage(wchar_t buffer[BUFFER_SIZE]) = 0`: чисто віртуальний метод для отримання повідомлення від сервера.