Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материала и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Чат на C++ с использование стандартных библиотек

Выполнил студент гр. 3331506/20102 Драчкова И. Ю.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2025

Оглавление

Введение

Целью данной курсовой работы является разработка простого чат-приложения на языке программирования C++ с использованием стандартных библиотек и средств работы с сокетами (WinSock API)**.** В ходе работы был реализован сервер, способный обрабатывать множественные подключения клиентов, а также клиентское приложение, обеспечивающее взаимодействие с сервером в режиме реального времени. Кроме того, предусмотрена поддержка комнат с паролями, что позволяет создавать изолированные каналы общения между пользователями.

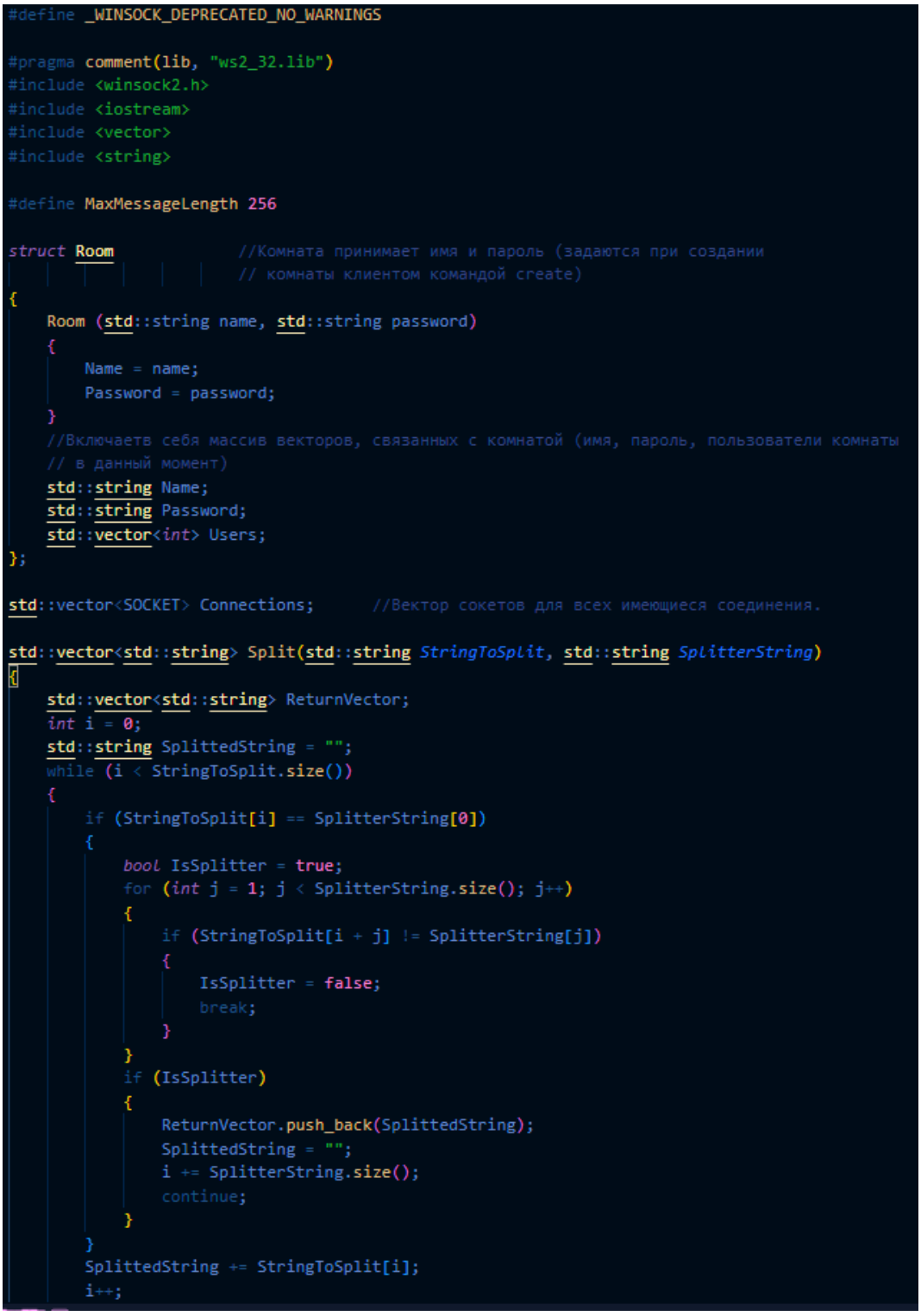
Основной задачей при реализации проекта было получение практических навыков сетевого программирования, понимание принципов клиент-серверного взаимодействия, а также работа с многопоточностью, обработкой сообщений и синхронизацией данных между подключёнными клиентами.

Результатом работы стало кроссплатформенное приложение, работающее в консольном режиме, не требующее сторонних библиотек и пригодное как для учебных целей, так и для демонстрации базовых принципов построения сетевых приложений.

1. Код программы сервера.

1.1. Код сервера.

На рисунке 1 представлен полный код программы Server с поясняющими комментариями. Ниже будет приведено развернутое описание основных функций кода.



```
#define _WINSOCK_DEPRECATED_NO_WARNINGS

#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h>
#include <iostream>
#include <vector>
#include <string>

#define MaxMessageLength 256

struct Room             //Комната принимает имя и пароль (задаются при создании
                        // комнаты клиентом командой create)
{
    Room (std::string name, std::string password)
    {
        Name = name;
        Password = password;
    }
    //Включаетв себя массив векторов, связанных с комнатой (имя, пароль, пользователи комнаты
    // в данный момент)
    std::string Name;
    std::string Password;
    std::vector<int> Users;
};

std::vector<SOCKET> Connections;      //Вектор сокетов для всех имеющиеся соединения.

std::vector<std::string> Split(std::string StringToSplit, std::string SplitterString)
{
    std::vector<std::string> ReturnVector;
    int i = 0;
    std::string SplittedString = "";
    while (i < StringToSplit.size())
    {
        if (StringToSplit[i] == SplitterString[0])
        {
            bool IsSplitter = true;
            for (int j = 1; j < SplitterString.size(); j++)
            {
                if (StringToSplit[i + j] != SplitterString[j])
                {
                    IsSplitter = false;
                    break;
                }
            }
            if (IsSplitter)
            {
                ReturnVector.push_back(SplittedString);
                SplittedString = "";
                i += SplitterString.size();
                continue;
            }
        }
        SplittedString += StringToSplit[i];
        i++;
```

```
    }
    ReturnVector.push_back(SplittedString);
    return ReturnVector;
}

template <class T>
void RemoveElementFromVectorByName(std::vector<T>& Vector, T Value)         //Стандартная функция, удаляющая первый элемнет
                                                                            // вектора (для корректного прочтения команд от
                                                                            // пользователя).
{
    for (auto it = Vector.begin(); it != Vector.end(); it++)
    {
        if (*it == Value)
        {
            Vector.erase(it);
            break;
        }
    }
}

std::vector<Room> Rooms;            //Вектор комнат для всех имеющихся комнат.

void MessageHandler(int UserId, char msg[MaxMessageLength], int* RoomId)    //Обработчик сообщений.
{
    std::vector<std::string> MessageVector = Split(std::string(msg), " ");
    std::string Message;

    if (MessageVector[0] == "!exit")                                        //Выход из комнаты.
    {
        RemoveElementFromVectorByName(Rooms[*RoomId].Users, UserId);
        Message = "Your exit room with name: " + Rooms[*RoomId].Name;
        *RoomId = -1;
        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
    }

    if (*RoomId != -1)                                                      //Если клиент находится в какой-то комнате.
    {
        for (int it : Rooms[*RoomId].Users)
        {
            if (UserId != it)
                send(Connections[it], msg, MaxMessageLength, NULL);
        }
        return;
    }

    if (MessageVector[0] == "ls")                                           //Показывает комнаты, которые уже созданы.
    {
        for (int i = 0; i < Rooms.size(); i++)
        {
            Message = Rooms[i].Name;
            send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        }
        return;
    }
```

```
    if (MessageVector[0] == "create" && MessageVector.size() >= 3)          //Создание новой комнаты.
    {
        Message = "You are created room: " + MessageVector[1];
        bool IsTakenName = false;
        for (auto it = Rooms.begin(); it != Rooms.end(); it++)
        {
            if ((*it).Name == MessageVector[1])
            {
                Message = "This room name alredy taken";
                IsTakenName = true;
                break;
            }
        }
        if (IsTakenName == false)
        {
            Room r(MessageVector[1], MessageVector[2]);
            Rooms.push_back(r);
        }

        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }

    if (MessageVector[0] == "create" && MessageVector.size() < 3)           //Заданы не все параметры для создания комнаты.
    {
        Message = "Wrong command. You have to specify room name and password\nCommand usage: create room_name room_password";
        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }

    if (MessageVector[0] == "remove" && MessageVector.size() >= 3)          //Удаление комнаты.
    {
        Message = "You are remove room: " + MessageVector[1];

        bool WasThisRoom = false;
        for (auto it = Rooms.begin(); it != Rooms.end(); it++)
        {
            if ((*it).Name == MessageVector[1])
            {
                WasThisRoom = true;
                if ((*it).Password == MessageVector[2])
                {
                    Rooms.erase(it);
                    break;
                }
                else
                    Message = "Wrong password";
            }
        }
```

```
        if (WasThisRoom == false)
            Message = "Wrong name";

        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }

    if (MessageVector[0] == "remove" && MessageVector.size() < 3)           //Недостаточно параметров для удаления комнаты
                                                                            // (чтобы не каждый клиент мог удалить любую комнату).
    {
        Message = "Wrong command. You have to specify room name and password\nCommand usage: remove room_name room_password";
        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }

    if (MessageVector[0] == "open" && MessageVector.size() >= 3)            //Присоединение к какой-то комнате.
    {
        Message = "Opened room: " + MessageVector[1];
        bool IsCorrectData = false;
        for (int i = 0; i < Rooms.size(); i++)
        {
            if (Rooms[i].Name == MessageVector[1] && Rooms[i].Password == MessageVector[2])
            {
                Rooms[i].Users.push_back(UserId);
                IsCorrectData = true;
                *RoomId = i;
                break;
            }
        }
        if (IsCorrectData == false)
            Message = "Wrong room name or password";

        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }

    if (MessageVector[0] == "open" && MessageVector.size() < 3)             //Недостаточно параметров для присоединения к комнате.
    {
        Message = "Wrong command. You have to specify room name and password\nCommand usage: open room_name room_password";
        send(Connections[UserId], Message.c_str(), MaxMessageLength, NULL);
        return;
    }
}

void ClientHandler(int i)       //Обрабатывает входные команды. i - номер клиента.
{
    std::cout << "Client Connected! id: " << i << std::endl;                //Указывает порядковый id клиента
                                                                            // (которые потом сохраняется и не сдвигается,
                                                                            // даже если один из предыдущих клиентов отсоединился).
    char msg[MaxMessageLength] = "Welcome. You are connected to server.";   //Приветственное сообщение клиенту.
    send(Connections[i], msg, MaxMessageLength, NULL);
```

```
    int ConnectionStatus;
    int RoomId = -1;

    //Приём сообщений от клиента.
    while (true)
    {
        ConnectionStatus = recv(Connections[i], msg, sizeof(msg), NULL);

        if (ConnectionStatus <= 0)        //Обрыв соединения.
        {
            std::cout << "Client disconnected. id: " << i << std::endl;
            if(RoomId != -1)
                RemoveElementFromVectorByName(Rooms[RoomId].Users, i);    //Удаление пользователя из комнаты.

            break;
        }

        MessageHandler(i, msg, &RoomId);
    }

    closesocket(i);
    Connections[i] = INVALID_SOCKET;
    if (i == Connections.size() - 1)
        Connections.pop_back();
    std::cout << Connections.size() << std::endl;
    return;
}

int main()
{
    //Создание IP адреса и порта.
    WSAData wsaData;
    if (WSAStartup(MAKEWORD(2, 1), &wsaData) != 0)
    {
        std::cout << "Error: Library initialization failure." << std::endl;
        exit(1);
    }

    SOCKADDR_IN addr;
    int sizeofaddr = sizeof(addr);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    addr.sin_port = htons(51111);
    addr.sin_family = AF_INET;

    //Сокет для получения запросов на подключение.
    SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL);
    bind(sListen, (SOCKADDR*)&addr, sizeof(addr));        //У сокета ip сервера, который указывался ранее.
    listen(sListen, SOMAXCONN);

    //Сокет, отвечающий за соединение с клиентом непосредственно.
    SOCKET newConnection;
```

```
    while (true)
    {
        newConnection = accept(sListen, (SOCKADDR*)&addr, &sizeofaddr);

        if (newConnection == INVALID_SOCKET)
        {
            std::cout << "Error: Client connection failure." << std::endl;
        }
        else
        {
            bool WasReusedSocket = false;
            for (int i = 0; i < Connections.size(); i++)
            {
                if (Connections[i] == INVALID_SOCKET)
                {
                    CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, (LPVOID)(i), NULL, NULL);
                    Connections[i] = newConnection;
                    WasReusedSocket = true;
                    std::cout << "Reused socket" << std::endl;
                    break;
                }
            }
            if (WasReusedSocket == false)
            {
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, (LPVOID)(Connections.size()), NULL, NULL);
                Connections.push_back(newConnection);
            }
        }

    }
}
```

Рисунок 1 – Код программы Server.

1.2. Описание основных функций сервера.

Основные функции кода сервера:

1. ***Split****(std::string, std::string)*

Разбивает строку на вектор подстрок по заданному разделителю. Используется для обработки входящих команд от клиента.

2. ***RemoveElementFromVectorByName****(std::vector<T>&, T)*

Удаляет первый найденный элемент из вектора. Применяется для удаления пользователя из комнаты или сокета из списка подключений.

3. ***MessageHandler****(int UserId, char msg[], int* RoomId)*

Обрабатывает сообщение, полученное от клиента:

- управление комнатами (*create, remove, open, ls, !exit*);
- пересылка сообщений другим пользователям в комнате;
- отправка ответов клиенту.

4. ***ClientHandler****(int i)*

Обрабатывает подключённого клиента в отдельном потоке:

- принимает сообщения от клиента;
- передаёт их в ***MessageHandler***;
- следит за отключением клиента и очищает данные после выхода.

5. ***main()***

Точка входа сервера:

- инициализирует библиотеку WinSock;
- настраивает серверный сокет (IP-адрес, порт);
- слушает подключения;
- создаёт новый поток (***ClientHandler***) на каждого подключённого клиента.

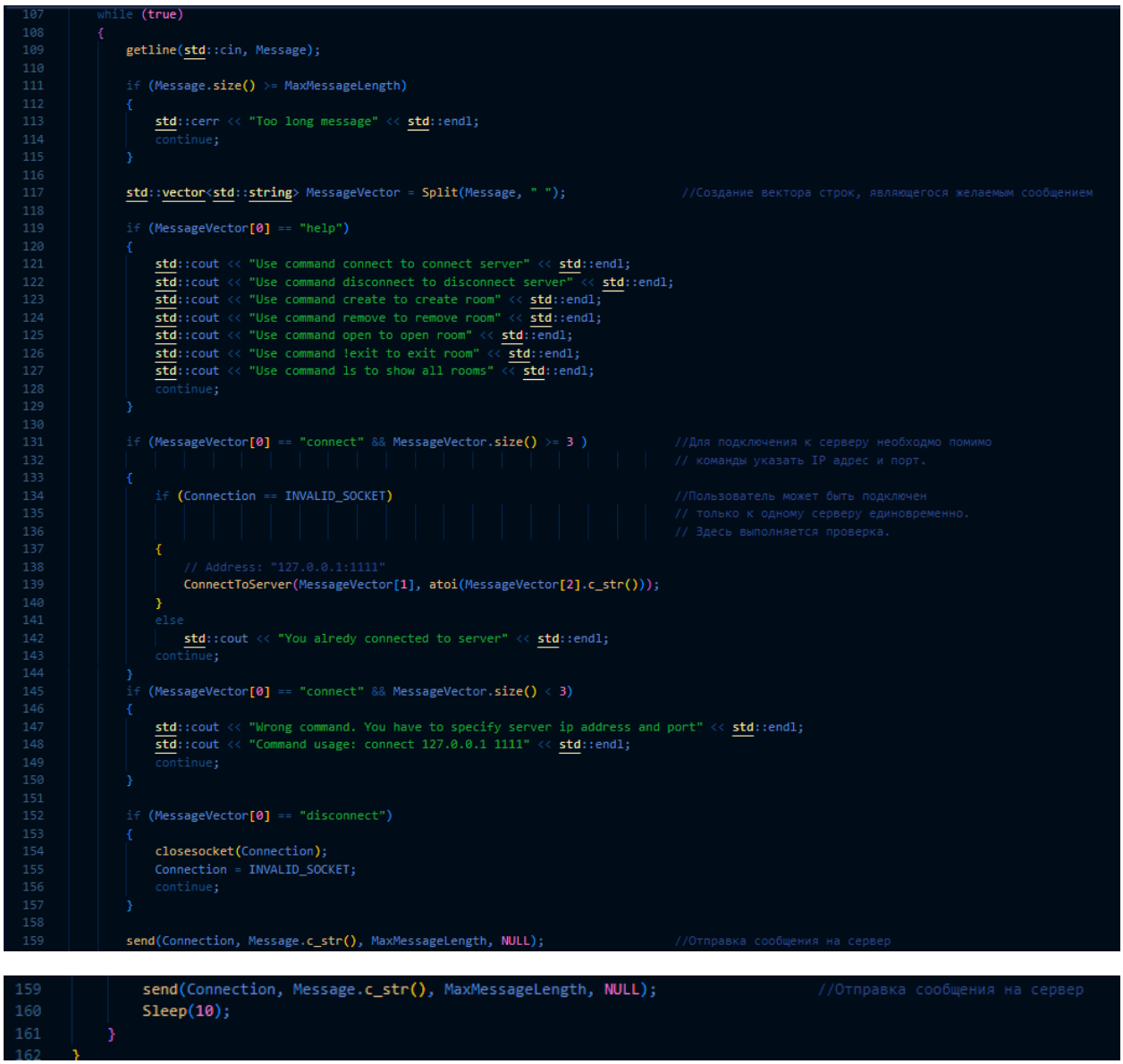
2. Код программы клиента.

2.1. Код клиента.

На рисунке 2 представлен полный код программы Client с поясняющими комментариями. Ниже будет приведено развернутое описание основных функций кода.

```cpp
#define _WINSOCK_DEPRECATED_NO_WARNINGS

#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h>
#include <iostream>
#include <string>
#include <vector>

#define MaxMessageLength 256                //Максимальная длина строки сообщения (для
                                            // более удобного разделения текста)
#define IsDebug true                        //Для автоматического подключения к серверу
                                            // при использовании кода на локальном компьютере

SOCKET Connection = INVALID_SOCKET;         //Подключение к сокету

std::vector<std::string> Split(std::string StringToSplit, std::string SplitterString)
{
    std::vector<std::string> ReturnVector;
    int i = 0;
    std::string SplittedString = "";
    while (i < StringToSplit.size())
    {
        if (StringToSplit[i] == SplitterString[0])
        {
            bool IsSplitter = true;
            for (int j = 1; j < SplitterString.size(); j++)
            {
                if (StringToSplit[i + j] != SplitterString[j])
                {
                    IsSplitter = false;
                    break;
                }
            }
            if (IsSplitter)
            {
                ReturnVector.push_back(SplittedString);
                SplittedString = "";
                i += SplitterString.size();
                continue;
            }
        }
        SplittedString += StringToSplit[i];
        i++;
    }
    ReturnVector.push_back(SplittedString);
    return ReturnVector;
}

void ClientHandler()                //Функция клиентской обработки
{
    char msg[256];
    int ConnectionStatus;
    while (true)
```

```cpp
    while (true)
    {
        ConnectionStatus = recv(Connection, msg, MaxMessageLength, NULL);   //Отвечает за то, что приходит.

        if (ConnectionStatus <= 0)                                          //Отключение сервера по каким-либо причинам.
        {
            std::cout << "Server disconnected" << std::endl;
            closesocket(Connection);
            Connection = INVALID_SOCKET;
            break;
        }

        std::cout << msg << std::endl;                                       //Вывод на экран ответа сервера.
    }
}

bool ConnectToServer(std::string ServerAddress, int Port)
{
    WSAData wsaData;
    if (WSAStartup(MAKEWORD(2, 1), &wsaData) != 0)
    {
        std::cerr << "Winsock init fail!" << std::endl;
        return false;
    }

    SOCKADDR_IN addr;                                       //Создание адреса
    int sizeofaddr = sizeof(addr);
    addr.sin_addr.s_addr = inet_addr(ServerAddress.c_str());
    addr.sin_port = htons(Port);
    addr.sin_family = AF_INET;

    Connection = socket(AF_INET, SOCK_STREAM, NULL);           //Сокет обеспечивает соединение с сервером

    if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr)) != 0)
    {
        std::cout << "Error: failed connect to server." << std::endl;
        return false;
    }

    CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, NULL, NULL, NULL);          //В одном потоке нельзя
                                                                                                // ждать сообщения и ввода с клавиатуры.
                                                                                                // Итого у клиента 2 потока.
    return true;
}

int main()
{
    std::string Message;

    std::cout << "Use command help to get information" << std::endl;

    if (IsDebug)
        ConnectToServer("127.0.0.1", 51111);          //Локальны адрес
```



```
    while (true)
    {
        getline(std::cin, Message);

        if (Message.size() >= MaxMessageLength)
        {
            std::cerr << "Too long message" << std::endl;
            continue;
        }

        std::vector<std::string> MessageVector = Split(Message, " ");              //Создание вектора строк, являющегося желаемым сообщением

        if (MessageVector[0] == "help")
        {
            std::cout << "Use command connect to connect server" << std::endl;
            std::cout << "Use command disconnect to disconnect server" << std::endl;
            std::cout << "Use command create to create room" << std::endl;
            std::cout << "Use command remove to remove room" << std::endl;
            std::cout << "Use command open to open room" << std::endl;
            std::cout << "Use command !exit to exit room" << std::endl;
            std::cout << "Use command ls to show all rooms" << std::endl;
            continue;
        }

        if (MessageVector[0] == "connect" && MessageVector.size() >= 3 )           //Для подключения к серверу необходмо помимо
                                                                                   // команды указать IP адрес и порт.
        {
            if (Connection == INVALID_SOCKET)                                      //Пользователь может быть подключен
                                                                                   // только к одному серверу единовременно.
                                                                                   // Здесь выполняется проверка.
            {
                // Address: "127.0.0.1:1111"
                ConnectToServer(MessageVector[1], atoi(MessageVector[2].c_str()));
            }
            else
                std::cout << "You alredy connected to server" << std::endl;
            continue;
        }
        if (MessageVector[0] == "connect" && MessageVector.size() < 3)
        {
            std::cout << "Wrong command. You have to specify server ip address and port" << std::endl;
            std::cout << "Command usage: connect 127.0.0.1 1111" << std::endl;
            continue;
        }

        if (MessageVector[0] == "disconnect")
        {
            closesocket(Connection);
            Connection = INVALID_SOCKET;
            continue;
        }

        send(Connection, Message.c_str(), MaxMessageLength, NULL);                 //Отправка сообщения на сервер
        send(Connection, Message.c_str(), MaxMessageLength, NULL);                 //Отправка сообщения на сервер
        Sleep(10);
    }
}
```

Рисунок 2 – Код программы Client.

2.2. Основные функции программы клиента.

Основные функции:

1. ***Split****(std::string, std::string)*

Разбивает введённую пользователем строку на части по пробелу. Используется для анализа команд (*connect, create, open* и др.).

2. ***ClientHandler()***

Функция, запущенная в отдельном потоке:

- слушает входящие сообщения от сервера через сокет;
- выводит их в консоль;
- завершает соединение при отключении сервера.

3. ***ConnectToServer****(std::string ServerAddress, int Port)*

Устанавливает соединение с сервером:

- создаёт и настраивает сокет;
- подключается по заданному IP-адресу и порту;
- запускает ***ClientHandler*** в отдельном потоке для приёма сообщений.

4. ***main()***

Основной цикл программы:

- выводит приветствие и справку по командам;
- в режиме отладки сразу подключается к локальному серверу (127.0.0.1);
- ожидает пользовательский ввод;
- проверяет команды (*connect*, *disconnect*, *help*, *create*, *remove*, *open*, *ls*, *!exit*);
- отправляет сообщения на сервер через *send()*.

Вывод.

В ходе выполнения курсовой работы была разработана и реализована простая клиент-серверная система обмена сообщениями (чат) на языке программирования C++ с использованием стандартной библиотеки сокетов Winsock. Основной целью проекта являлось закрепление навыков сетевого программирования, работы с многопоточностью, а также реализация базового функционала для взаимодействия нескольких клиентов через сервер.

Разработанный сервер способен:

- обрабатывать подключения нескольких клиентов;
- управлять созданием и удалением комнат;
- пересылать сообщения между клиентами, находящимися в одной комнате.

Клиентская часть позволяет пользователю:

- подключаться к серверу;
- взаимодействовать с чат-комнатами;
- отправлять и получать сообщения в реальном времени.

Полученные в процессе реализации знания и навыки могут быть использованы для более сложных сетевых приложений, включая мессенджеры, игры и распределённые системы.

Список литературы.

1. Бьерн Страуструп. *Язык программирования C++. Базовые принципы и практика.* — СПб: Питер, 2021.
2. Стивен Прата. *C++. Лекция и практика.* — СПб: Питер, 2020.
3. MSDN Documentation – Winsock Reference.
4. Шилдт Г. *C++ для начинающих*. — М.: Вильямс, 2019.
5. Официальный сайт Microsoft Docs — https://learn.microsoft.com/
6. Иванов С. А., Петров А. В. *Реализация клиент-серверного взаимодействия в C++ с использованием библиотеки WinSock* // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. — 2022. — №3. — С. 45–51.
7. Сидоров И. И., Малова Е. Н. *Применение сокетов в разработке сетевых приложений на языке C++* // Современные проблемы науки и образования. — 2021. — №6. — URL: https://science-education.ru
8. Романов П. Н. *Методы организации обмена данными между клиентом и сервером на C++* // Информационные технологии и вычислительные системы. — 2020. — №4. — С. 67–73.
9. Кузнецов В. А., Лебедев А. С. *Программирование сетевых взаимодействий в операционной системе Windows средствами WinAPI* // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. — 2021. — №2. — С. 91–98.
10. Федорова Т. Е., Белова М. Ю. *Применение многопоточности в клиент-серверных приложениях на C++* // Труды Международной конференции «Информационные технологии и системы». — 2022. — С. 112–118.