Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №5

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Студент: Ющук И.А.

Преподаватель: Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc181894300)

[2 Описание функций программы 4](#_Toc181894301)

[2.1 Ожидание подключений к серверу 4](#_Toc181894302)

[2.2 Отображение сообщений на сервере 4](#_Toc181894303)

[2.3 Отправка сообщений 4](#_Toc181894304)

[2.4 Передача данных по именованному каналу 5](#_Toc181894305)

[Заключение 6](#_Toc181894306)

[Список использованных источников 7](#_Toc181894307)

[Приложение А (обязательное) Исходный код программы 8](#_Toc181894308)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Упрощенный чат для нескольких пользователей (в целом подобно заданию из темы «Обмен данными») с использованием сетевых сокетов.

Транспортные протоколы: TCP или UDP.

Архитектура: централизованная (выделенный процесс-сервер и процессы-клиенты) или децентрализованная (процессы-клиенты с «серверными» функциями).

Сервер: создание сокета для приема соединений или отдельных сообщений; прием и временное хранение сообщений; передача сообщений адресно одному или нескольким клиентам; поддержание списка актуальных клиентов.

Клиент: обнаружение сервера и соединение с ним; ввод пользовательских сообщений и передача их серверу либо напрямую соответствующему клиенту; прием и отображение сообщений.

Потребуется разработка структуры передаваемых сообщений (как минимум, адрес и тело сообщения) и порядка обмена (протокола прикладного уровня).

Опционально: в децентрализованной системе маршрутизация сообщений (возможность трансляции сообщений по цепочке участников).

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– функция для ожидания подключений к серверу;

– функция для отображения сообщений от клиентов на сервере;

– функция для отправки сообщений на сервер;

– функция для отображения сообщений от других пользователей на клиенте.

## **2.1 Ожидание подключений к серверу**

При запуске сервера создается TCP-сокет с использованием функции socket() [1]. Этот сокет используется для приёма подключений от клиентов. Сервер связывает сокет с определённым IP-адресом и портом с помощью функции bind() [2]. Это указывает, на каком адресе и порте сервер будет ожидать подключения клиентов. Далее сервер переводит сокет в режим прослушивания с помощью listen() [3]. Эта функция ставит сокет в состояние ожидания входящих соединений от клиентов. В функции accept() [4] сервер принимает входящее соединение от клиента. Эта функция блокирует выполнение программы до тех пор, пока клиент не попытается подключиться. Как только клиент подключается, accept() создаёт новый сокет для этого соединения и возвращает его дескриптор. Это позволяет серверу обслуживать несколько клиентов параллельно (рисунок 2.1).

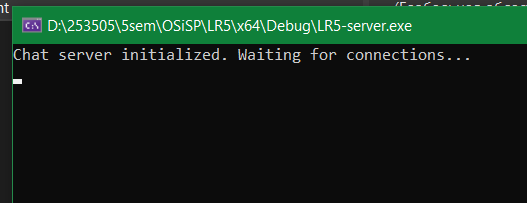


Рисунок 2.1 – Ожидание подключения к серверу

## **2.2 Отображение сообщений на сервере**

На сервере также присутствует функция для отображения сообщений, пришедших от клиента с указанием отправителя (рисунок 2.2).

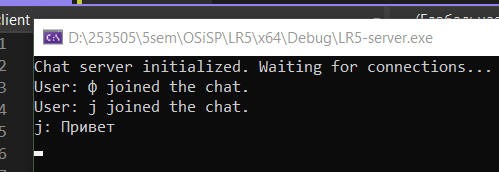


Рисунок 2.2 – Сообщения на сервере

## **2.3 Отправка сообщений**

На стороне клиента пользователь вводит своё сообщение в консоли, и программа сохраняет это сообщение вместе с именем пользователя в структуру ChatMessage (рисунок 2.3) [3]. После ввода сообщения клиент отправляет структуру ChatMessage на сервер с использованием функции send() [5]. Она передаёт данные через TCP-сокет, установив соединение с сервером. Сервер получает данные от подключённых клиентов с помощью функции recv() [6], которая принимает данные из сокета, связанного с каждым клиентом.

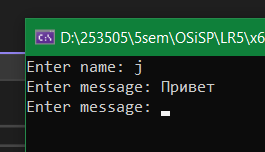


Рисунок 2.3 – Ввод сообщения

## **2.4 Передача данных по именованному каналу**

На клиенте также присутствует функция для отображения сообщений, присланных от других пользователей (рисунок 2.4).

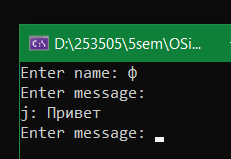


Рисунок 2.4 – Отображение сообщения от других пользователей

ЗАКЛЮЧЕНИЕ

В данной задаче реализован упрощённый чат. Программа построена на протоколе TCP, что обеспечивает надёжное соединение между клиентами и сервером, а также корректную доставку сообщений.

Сервер ожидает подключения клиентов, создаёт для каждого поток и пересылает полученные сообщения другим клиентам, обеспечивая параллельное обслуживание. Клиенты подключаются к серверу, отправляют и получают сообщения, используя структуру ChatMessage для упорядоченного обмена данными.

Реализация демонстрирует основные принципы сетевого программирования, а именно работу с TCP-сокетами, многопоточность, управление соединениями и передачу данных. Такой чат может служить основой для более сложных сетевых приложений и практическим примером сетевой коммуникации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] socket function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket.

[2] bind function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind.

[3] listen function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-listen.

[4] accept function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-accept.

[5] send function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-send.

[6] recv function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-recv.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

#include <WinSock2.h>

#include <Ws2tcpip.h>

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <algorithm>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 8000

using namespace std;

struct ChatMessage {

char author[50];

char text[256];

};

struct ConnectionInfo {

SOCKET conn\_socket;

string username;

};

vector<ConnectionInfo> active\_connections;

mutex connection\_mutex;

void SendToAllClients(const ChatMessage& message, const string& exclude\_user) {

lock\_guard<mutex> lock(connection\_mutex);

for (const auto& client : active\_connections) {

if (client.username != exclude\_user) {

send(client.conn\_socket, reinterpret\_cast<const char\*>(&message), sizeof(message), 0);

}

}

}

void HandleClient(SOCKET client\_conn\_socket) {

ChatMessage received\_msg;

int received\_bytes = recv(client\_conn\_socket, reinterpret\_cast<char\*>(&received\_msg), sizeof(received\_msg), 0);

if (received\_bytes <= 0) {

closesocket(client\_conn\_socket);

return;

}

string current\_username = received\_msg.author;

{

lock\_guard<mutex> lock(connection\_mutex);

active\_connections.push\_back({ client\_conn\_socket, current\_username });

cout << "User: " << current\_username << " joined the chat.\n";

}

while (true) {

int received\_bytes = recv(client\_conn\_socket, reinterpret\_cast<char\*>(&received\_msg), sizeof(received\_msg), 0);

if (received\_bytes <= 0) break;

cout << received\_msg.author << ": " << received\_msg.text << endl;

SendToAllClients(received\_msg, current\_username);

}

closesocket(client\_conn\_socket);

{

lock\_guard<mutex> lock(connection\_mutex);

active\_connections.erase(remove\_if(active\_connections.begin(), active\_connections.end(),

[&](const ConnectionInfo& client) {

return client.conn\_socket == client\_conn\_socket;

}), active\_connections.end());

cout << "User: " << current\_username << " left the chat.\n";

}

}

int main() {

WSADATA ws\_data;

WSAStartup(MAKEWORD(2, 2), &ws\_data);

SOCKET listen\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

sockaddr\_in server\_info;

server\_info.sin\_family = AF\_INET; //IPv4

server\_info.sin\_port = htons(PORT); //PORT

server\_info.sin\_addr.s\_addr = INADDR\_ANY; // для принятия трафика с любых IP

bind(listen\_socket, (sockaddr\*)&server\_info, sizeof(server\_info));

listen(listen\_socket, SOMAXCONN);

cout << "Chat server initialized. Waiting for connections...\n";

while (true) {

SOCKET new\_client\_socket = accept(listen\_socket, nullptr, nullptr);

thread(HandleClient, new\_client\_socket).detach();

}

closesocket(listen\_socket);

WSACleanup();

return 0;

}