Министерство науки и высшего образования Российской Федерации Пензенский государственный университет

Кафедра «Вычислительная техника»

Выполнила:

студентка группы 21ВВП1

Алёшина А.В.

Приняли:

Егоров В.Ю.

Федюнин Р.Н.

Пенза 2024

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Основы операционных систем»

на тему «Система программ обмена текстовыми сообщениями»

**Оглавление**

[Введение 3](#_Toc176867264)

[1. Постановка задачи 4](#_Toc176867265)

[2. Теоретическая часть задания 5](#_Toc176867266)

[3. Описание алгоритма работы 6](#_Toc176867267)

[4. Описание программы 9](#_Toc176867268)

[4.1. Клиент 9](#_Toc176867269)

[4.2. Сервер 10](#_Toc176867270)

[5. Описание работы программы 12](#_Toc176867271)

[Заключение 16](#_Toc176867272)

[Список используемых источников 17](#_Toc176867273)

[Приложение A. Листинг программы 18](#_Toc176867274)

[Приложение A.1. Файл «Server.cpp» для Сервера 18](#_Toc176867275)

[Приложение A.2. Файл «Program.cs» для Клиента 22](#_Toc176867276)

[Приложение A.3. Файл «Form1.cs» для Клиента 25](#_Toc176867277)

[Приложение A.4. Файл « Form2.cs» для Клиента 27](#_Toc176867278)

# Введение

В наше время возросла потребность в мгновенном общении, что привело к увеличению предложения различных услуг в сфере мессенджеров. Эти современные коммуникационные платформы не только обеспечивают обмен текстовыми сообщениями через интернет, но также позволяют передавать файлы, выражать эмоции при помощи разнообразных смайликов, в том числе анимированных изображений и видеороликов. Групповые чаты стали нормой, обеспечивая коммуникацию между людьми, работающими над общими задачами, независимо от их расположения на планете. Появление голосовых комнат в мессенджерах для семейных обсуждений или деловых встреч стало еще одним важным этапом в развитии этой технологии.

Архитектура клиент-сервер представляет собой распределенную систему, где задачи и сетевая нагрузка распределяются между поставщиками услуг (серверами) и их потребителями (клиентами). Взаимодействие между клиентами и серверами происходит через компьютерные сети и может включать как различные физические устройства, так и программное обеспечение. Пример простой технологии клиент-сервера заключается в том, что пользователь делает запрос (например, поиск информации в Google), а сервер предоставляет ответ в виде списка сайтов, соответствующих запросу.

Целью данного курсового проекта является разработка системы распределенного доступа к текстовому документу, состоящей из сервера и клиентов, взаимодействующих по сети. Совокупность интернет-протоколов обеспечивает сквозную передачу данных, определяя, как данные должны быть упакованы, обработаны, переданы, маршрутизированы и приняты. Эта функциональность организована в четыре слоя абстракции: уровень связи, интернет-уровень, транспортный уровень и прикладной уровень, которые классифицируют протоколы в зависимости от их вовлеченности в сети.

# Постановка задачи

В курсовой работе необходимо разработать систему программ клиент-серверной архитектуры, позволяющей осуществлять параллельную передачу сообщений между клиентами посредством работы с TCP/IP протоколами.

Необходимо создать сервер и клиент. Клиенту обрабатывать команды, поступающие от сервера. Сервер в этот момент синхронизирует клиентов, собирает и обрабатывает данные и выдает их клиентам.

Функции клиента:

* Интерфейс пользователя
* Взаимодействие с сервером

Функции сервера:

* Синхронизация клиентских приложений
* Отслеживание подключений и отключений клиентов
* Доставка сообщений клиентам

Операционная система – Windows.

Язык программирования – C++/C#.

Среда программирования: Microsoft Visual Studio 2022.

# Теоретическая часть задания

"Клиент-серверная архитектура" представляет собой организацию вычислительных или сетевых систем, где задачи и сетевая нагрузка распределены между поставщиками услуг, именуемыми серверами, и их потребителями, которые называются клиентами. Фактически клиент и сервер представляют собой программное обеспечение, обычно размещенное на различных вычислительных устройствах, и взаимодействующее друг с другом через вычислительные сети с использованием сетевых протоколов. Однако они также могут быть размещены на одной машине. В случае поступления нескольких запросов одновременно, они упорядочиваются в очередь и обрабатываются сервером последовательно. Иногда запросы могут иметь приоритеты, и запросы с более высоким приоритетом выполняются в первую очередь.

Интернет, несмотря на свою обширность и сложность, на самом базовом уровне представляет собой взаимодействие между различными компьютерами, не ограничиваясь только персональными. Это взаимодействие осуществляется с использованием сетевых протоколов передачи данных - набора правил, определяющих порядок и характер передачи информации в конкретных случаях. Программы в Интернете взаимодействуют между собой с использованием протоколов.

Протокол — это набор правил и соглашений, применяемых при передаче данных. Таким образом, каждая программа, предназначенная для работы в сети, должна следовать определенным правилам для приема и передачи данных. Основным протоколом в сети Интернет является протокол TCP/IP, который представляет собой стек протоколов TCP и IP. TCP обеспечивает надежную передачу данных и следит за их целостностью, в то время как IP отвечает за маршрутизацию передачи данных. Часто протокол TCP используется более сложными протоколами.

# Описание алгоритма работы

Разработанная программа основывается на вышеописанных протоколах. Она состоит из клиента и сервера. Сначала запускаем сервер и создаем сокет. Устанавливаем локальную точку для прослушивания подключений. Запускаем прослушивание входящих подключений и принимаем новых клиентов в бесконечном цикле. После того, как клиент был принят, он передается на обработку новой нить в функцию обработки клиентов. Таким образом, для каждого нового клиента создается нить на стороне сервера, которая общается только со своим клиентом. На этом установление соединения между клиентом и сервером заканчивается. После установления соединения сервер в бесконечном цикле ожидает команды от клиента.

Запускаем клиент и подключаемся к серверу. Создаем новую нить для приема сообщения и ожидаем сообщения в бесконечном цикле.

Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание

Рисунок 1 - UML- диаграмма протокола взаимодействия клиента и сервера.



Рисунок 2 - UML-диаграмма вариантов использования.

Изображение выглядит как текст, снимок экрана, визитная карточка, Шрифт

Автоматически созданное описание

Рисунок 3 – UML-диаграмма развертывания.

# Описание программы

## Клиент

Программа клиента состоит из нескольких основных модулей:

private void LoginBtn\_Click(object sender, EventArgs e) – метод кнопки подключения к серверу пользовательского интерфейса клиента (подключение именно к серверу происходит в момент запуска приложения, метод же отвечает за подключение пользователя с помощью логина).

public static void ReceiveMessages(object obj) – нить для прослушивания сообщений от сервера.

private void SendBtn\_Click(object sender, EventArgs e) – метод кнопки отправки сообщения серверу пользовательского интерфейса клиента.

private void Form1Closing(object sender, FormClosingEventArgs e) – метод слушатель закрытия окна пользовательского интерфейса клиента и отключения клиента от сервера.

Ниже представлена схема по организации работы клиента по протоколу TCP/IP.

Изображение выглядит как диаграмма, План, Технический чертеж, текст

Автоматически созданное описание

Рисунок 4 - Общая схема работы клиента.

## Сервер

Программа сервера использует winsock2, сначала создаётся сокет сервера, затем при подключении клиента создаётся нить void HandleClient(User\* user), которая обрабатывает все запросы со стороны клиент (сервер ожидает в бесконечно цикле подключение пользователей).

Данные пользователя (логин и сокет) хранятся в векторе, при отключении пользователя сокет закрывается, затем данные удаляются, это всё происходит в рамках нити void HandleClient(User\* user).

В нити принимается ряд команд:

"message/" <сообщение> - сервер рассылает сообщение всем пользователям, оно отобразится в общем чате.

"message\_for/" <логин принимающего> <логин отправителя> <сообщение> - сервер отправляет сообщение двум клиентам “принимающему” и “отправителю” (они отобразятся как личные сообщения).

"login/" <логин> - проверяет есть ли пользователь с таким же логином на сервере, отправляет всем остальным клиентам информацию о том, что вошёл новый пользователь.

"getusers/" - клиент получает список пользователей онлайн.

"disconnect/" - закрывает сокет отключившегося пользователя, сообщает другим клиентам об отключение.

Ниже представлена схема по организации работы сервера по протоколу TCP/IP.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Рисунок 5 - Общая схема работы сервера.

# Описание работы программы

Результаты работы программы изображены на рисунках 5-11.

Сначала нужно запустить сервер (Рисунок 5).

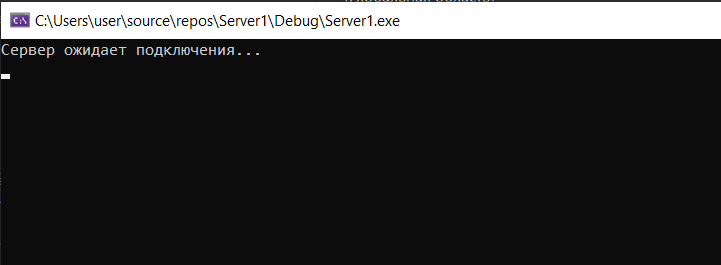


Рисунок 6 - Окно серверного приложения.

Далее необходимо запустить клиентское приложение (Рисунок 7) и ввести имя, а за затем нажать кнопку «Войти» или Enter. В случае успешного подключения откроется другое окно с чатом, а также в списке слева отобразятся имена пользователей онлайн (если они есть).

При подключении нового клиента в окне серверного приложения отобразится соответствующее сообщение (Рисунок 6).

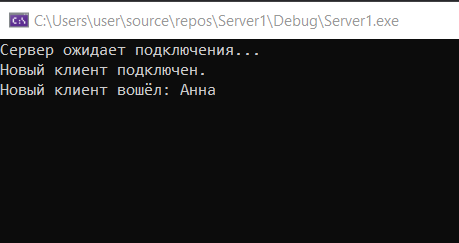


Рисунок 7 - Окно серверного приложения.

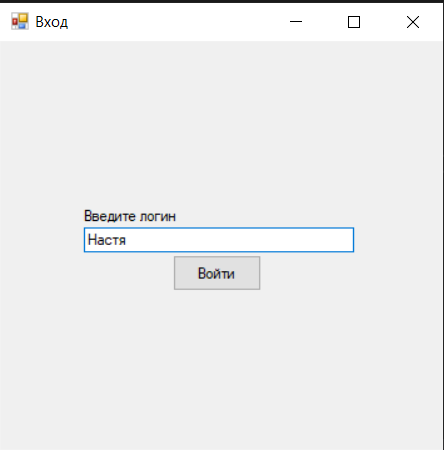


Рисунок 8 - Окно клиентского приложения.

Для того чтобы начать обмен сообщениями между пользователями, нужно ввести текст внутри текстового пространства снизу и нажать кнопку «Оправить» на интерфейсе или Enter. После чего набранное сообщение будет отправлено всем клиентам и будет отображаться в самом большом текстовом окне.

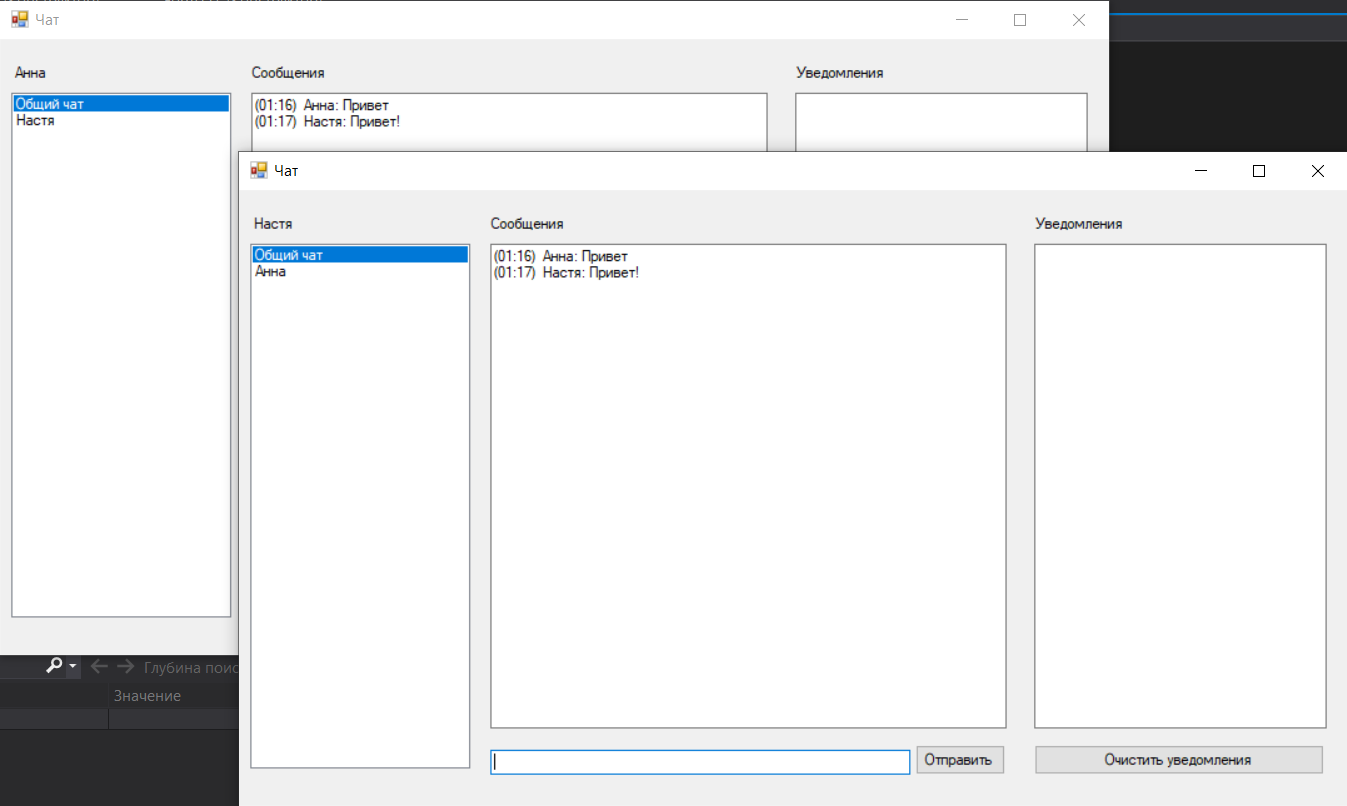


Рисунок 9 – Окно клиентского приложения с отправленным сообщением.

Все сообщения пользователей и действия сервера отображаются в окне сервера (Рисунок 9).

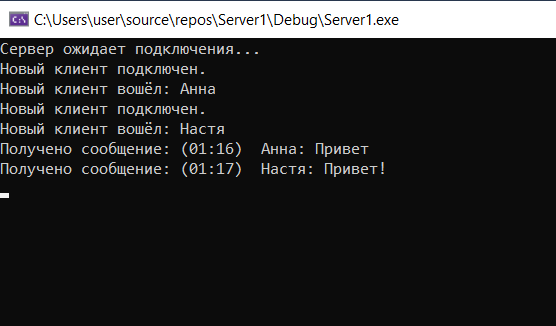


Рисунок 10 – Окно серверного приложения.

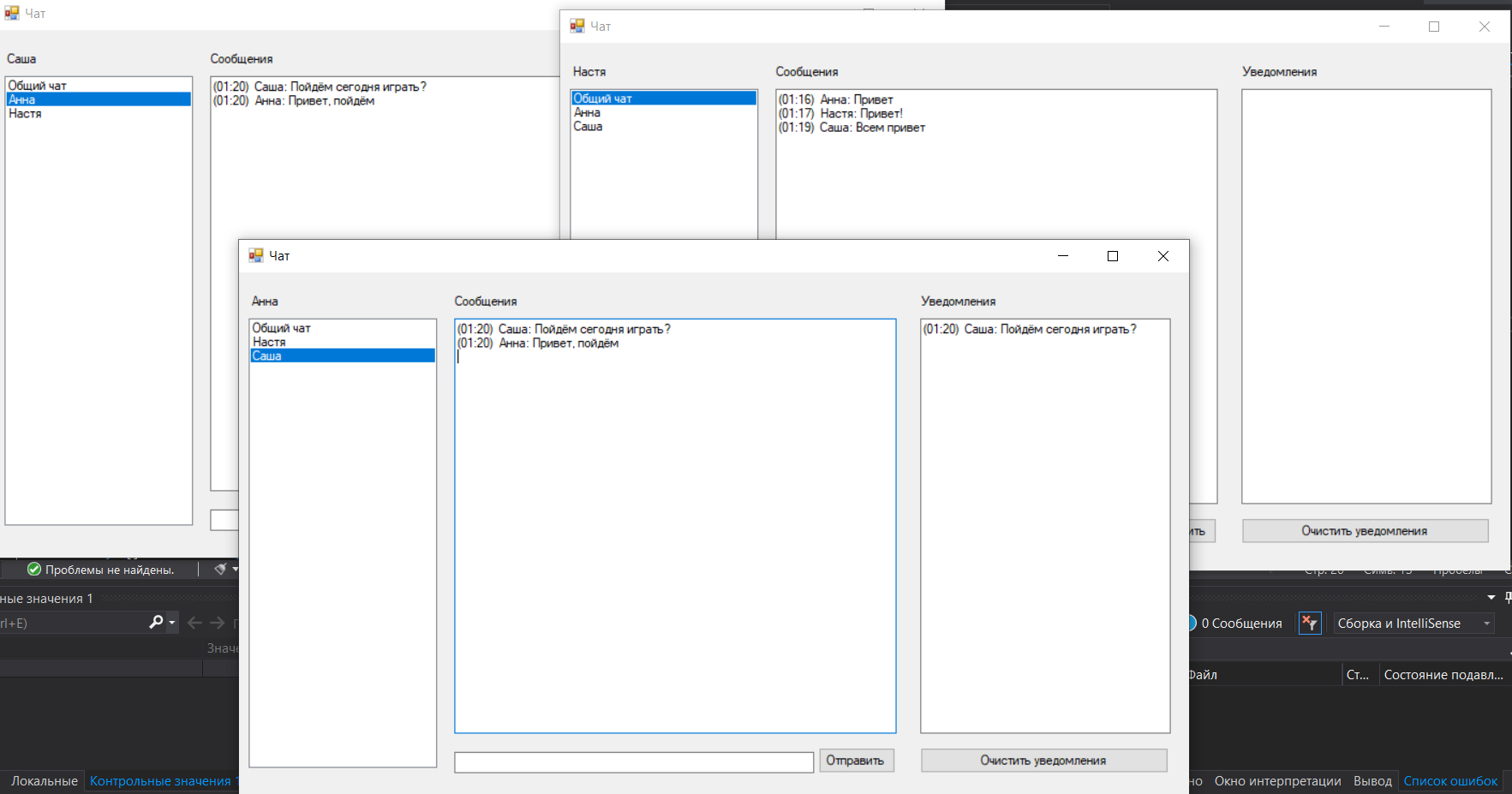


Рисунок 11 – Окно клиентского приложения.

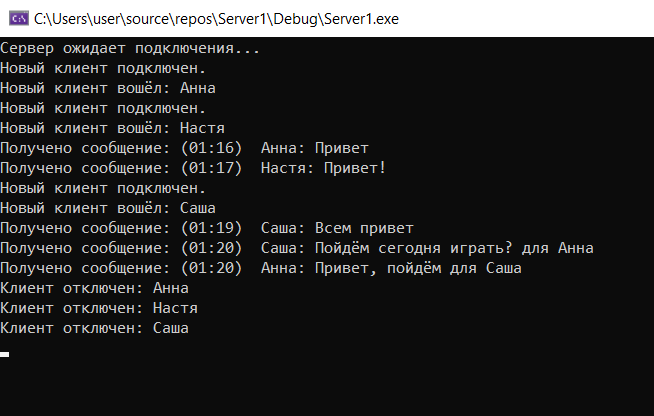


Рисунок 12 – Окно серверного приложения.

# Заключение

При выполнении данного курсового проекта были изучены принципы работы протоколов TCP/IP, разработка клиента и сервера. Организовали синхронную работы программы клиент-серверной архитектуры. Усвоили механизм обработки событий в формах и пользовательских инструментах.

В результате выполнения данной курсовой работы была разработана система программ клиент-серверной архитектуры, позволяющей осуществлять параллельную передачу сообщений между клиентами с помощью сервера посредством работы с TCP/IP протоколами.

Программа написана на языке C# и C++ с использование среды программирования Visual Studio 2022.

# Список используемых источников

1. [METANIT.COM](https://metanit.com/) Сайт о программировании [Электронный ресурс]. Режим доступа: <http://www.cyberforum.ru/>, свободный (Дата обращения 24.12.2023).
2. learn.microsoft.com Сайт о программировании [Электронный ресурс]. Режим доступа: <https://learn.microsoft.com/en-us/dotnet/csharp/>, свободный (Дата обращения 24.12.2023).
3. en.cppreference.com Сайт о программировании [Электронный ресурс]. Режим доступа: <https://en.cppreference.com/w/>, свободный (Дата обращения 24.12.2023).

# Приложение A. Листинг программы

## Приложение A.1.

## Файл «Server.cpp» для Сервера

#include <iostream>

#include <winsock2.h>

#include <vector>

#include <thread>

#include <mutex>

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

struct User {

SOCKET clientSocket;

char login[32] = "";

};

std::vector<User\*> users;

std::mutex usersMutex;

void HandleClient(User\* user) {

char buffer[1024];

while (true) {

memset(buffer, 0, sizeof(buffer));

int bytesRead = recv(user->clientSocket, buffer, sizeof(buffer), 0);

std::string command(buffer);

if (command.find("message/") == 0) {

std::string message = command.substr(8);

std::cout << "Получено сообщение: " << message << std::endl;

send(user->clientSocket, buffer, bytesRead, 0);

// Рассылка сообщения всем подключенным клиентам, кроме отправителя

std::lock\_guard<std::mutex> lock(usersMutex);

for (const User\* otherUser : users) {

if (otherUser->clientSocket != user->clientSocket) {

send(otherUser->clientSocket, buffer, bytesRead, 0);

}

}

}

else if (command.find("message\_for/") == 0) {

const char\* prefix = "message\_for/";

const size\_t prefixLength = strlen(prefix);

const char\* dataStart = buffer + prefixLength;

char request[1024];

strcpy\_s(request, dataStart);

size\_t lastPipeIndex = command.find\_last\_of('/');

size\_t firstPipeIndex = command.find('/');

size\_t secondPipeIndex = command.find('/', firstPipeIndex + 1);

std::string login = command.substr(command.find('/') + 1, secondPipeIndex - command.find('/') - 1);

std::string message = command.substr(lastPipeIndex + 1);

std::cout << "Получено сообщение: " << message << " для " << login << std::endl;

std::lock\_guard<std::mutex> lock(usersMutex);

for (const User\* otherUser : users) {

if (otherUser->login == login) {

send(otherUser->clientSocket, buffer, bytesRead, 0);

}

}

sprintf\_s(buffer, "%s%s", "message\_from/", request);

send(user->clientSocket, buffer, bytesRead + 1, 0);

}

else if (command.find("login/") == 0) {

std::string loginInfo = command.substr(6);

bool loginExists = false;

for (const User\* otherUser : users) {

if (strcmp(otherUser->login, loginInfo.c\_str()) == 0) {

loginExists = true;

break;

}

}

std::lock\_guard<std::mutex> lock(usersMutex);

if (loginExists) {

send(user->clientSocket, "login\_error/", strlen("login\_error/"), 0);

}

else {

strncpy\_s(user->login, loginInfo.c\_str(), sizeof(user->login) - 1);

std::cout << "Новый клиент вошёл: " << user->login << std::endl;

send(user->clientSocket, "login\_success/", strlen("login\_success/"), 0);

sprintf\_s(buffer, "%s%s", "getusers/", user->login);

int len\_buffer = strlen(buffer);

for (const User\* otherUser : users) {

if (otherUser->clientSocket != user->clientSocket) {

if (strlen(otherUser->login) != 0) {

send(otherUser->clientSocket, buffer, len\_buffer, 0);

}

}

}

}

}

else if (command.find("getusers/") == 0) {

std::lock\_guard<std::mutex> lock(usersMutex);

for (const User\* otherUser : users) {

if (otherUser->clientSocket != user->clientSocket) {

if (strlen(otherUser->login) != 0) {

sprintf\_s(buffer, "%s%s", "getusers/", otherUser->login);

send(user->clientSocket, buffer, strlen(buffer), 0);

}

}

}

}

else if (command.find("disconnect/") == 0) {

std::lock\_guard<std::mutex> lock(usersMutex);

closesocket(user->clientSocket);

for (const User\* otherUser : users) {

if (otherUser->clientSocket != user->clientSocket) {

if (strlen(otherUser->login) != 0) {

sprintf\_s(buffer, "%s%s", "disconnect/", user->login);

send(otherUser->clientSocket, buffer, strlen(buffer), 0);

}

}

}

std::cout << "Клиент отключен: " << user->login << std::endl;

// Удаление из вектора

auto it = std::find\_if(users.begin(), users.end(),

[user](const User\* u) { return u->clientSocket == user->clientSocket; });

if (it != users.end()) {

users.erase(it);

}

break;

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

WSADATA wsaData;

WSAStartup(MAKEWORD(2, 2), &wsaData);

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

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(8888);

bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

listen(serverSocket, 5);

std::cout << "Сервер ожидает подключения..." << std::endl;

while (true) {

User\* newUser = new User;

newUser->clientSocket = accept(serverSocket, NULL, NULL);

std::lock\_guard<std::mutex> lock(usersMutex);

users.push\_back(newUser);

std::cout << "Новый клиент подключен." << std::endl;

std::thread(HandleClient, newUser).detach();

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

## Приложение A.2. Файл «Program.cs» для Клиента

using System;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace ClientApp

{

internal static class Program

{

public static TcpClient tcpClient;

public static NetworkStream networkStream;

public static Thread receiveThread;

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

tcpClient = new TcpClient("127.0.0.1", 8888);

networkStream = tcpClient.GetStream();

receiveThread = new Thread(ReceiveMessages);

receiveThread.Start(networkStream);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

public static void ReceiveMessages(object obj)

{

networkStream = (NetworkStream)obj;

byte[] buffer = new byte[1024];

while (true)

{

int bytesRead = networkStream.Read(buffer, 0, buffer.Length);

string receivedMessage = Encoding.GetEncoding("windows-1251").GetString(buffer, 0, bytesRead);

if (receivedMessage.StartsWith("message/"))

{

string message = receivedMessage.Substring(8);

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

form1Instace.form2.generalChat += (message + Environment.NewLine);

form1Instace.form2.UpdateMessages();

}

}

else if (receivedMessage.StartsWith("message\_for/"))

{

int firstPipeIndex = receivedMessage.IndexOf('/');

int secondPipeIndex = receivedMessage.IndexOf('/', firstPipeIndex + 1);

int thirdPipeIndex = receivedMessage.IndexOf('/', secondPipeIndex + 1);

string login = receivedMessage.Substring(secondPipeIndex + 1, thirdPipeIndex - secondPipeIndex - 1);

string message = receivedMessage.Substring(thirdPipeIndex + 1);

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

form1Instace.form2.UpdateMessage(login, message);

form1Instace.form2.UpdateMessages();

}

}

else if (receivedMessage.StartsWith("message\_from/"))

{

int firstPipeIndex = receivedMessage.IndexOf('/');

int secondPipeIndex = receivedMessage.IndexOf('/', firstPipeIndex + 1);

int thirdPipeIndex = receivedMessage.IndexOf('/', secondPipeIndex + 1);

string login = receivedMessage.Substring(firstPipeIndex + 1, secondPipeIndex - firstPipeIndex - 1);

string message = receivedMessage.Substring(thirdPipeIndex + 1);

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

form1Instace.form2.UpdateMessage(login, message);

form1Instace.form2.UpdateMessages();

}

}

else if (receivedMessage.StartsWith("getusers/"))

{

string login = receivedMessage.Substring(9);

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

form1Instace.form2.DisplayUser(login);

form1Instace.form2.messages.Add(new UserMessage { Login = login, Message = "" });

}

}

else if (receivedMessage.StartsWith("disconnect/"))

{

string login = receivedMessage.Substring(11);

Form1 form1Instace = Form1.Instance;

form1Instace?.form2.RemoveUser(login);

}

else if (receivedMessage.StartsWith("login\_error/"))

{

MessageBox.Show($"Логин уже занят", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else if (receivedMessage.StartsWith("login\_success/"))

{

Form1 form1Instace = Form1.Instance;

form1Instace?.CreateForm2();

}

}

}

public static void SendMessage(string message)

{

try

{

byte[] buffer = Encoding.GetEncoding("windows-1251").GetBytes(message);

networkStream.Write(buffer, 0, buffer.Length);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка отправки сообщения: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

## Приложение A.3.

## Файл «Form1.cs» для Клиента

using System;

using System.Windows.Forms;

namespace ClientApp

{

public partial class Form1 : Form

{

private static Form1 instance;

public Form1()

{

InitializeComponent();

this.Text = "Вход";

StartPosition = FormStartPosition.CenterScreen; // Устанавливаем центральное положение формы

instance = this;

this.FormClosing += Form1Closing;

loginBox.KeyPress += LoginBox\_KeyPress;

}

private void LoginBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == (char)Keys.Enter)

{

LoginBtn\_Click(sender, e);

}

}

public static Form1 Instance

{

get { return instance; }

}

public Form2 form2;

public string login;

private void LoginBtn\_Click(object sender, EventArgs e)

{

login = loginBox.Text;

if (string.IsNullOrWhiteSpace(login))

{

MessageBox.Show("Пожалуйста, введите логин.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (login.Contains("/"))

{

MessageBox.Show("Логин содержит запрещенный символ '/'.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

Program.SendMessage("login/" + login);

}

public void CreateForm2()

{

this.BeginInvoke((MethodInvoker)delegate

{

form2 = new Form2();

form2.Form2Closed += Form2\_FormClosed;

this.Hide();

form2.Show();

});

}

private void Form1Closing(object sender, FormClosingEventArgs e)

{

Program.SendMessage("disconnect/");

if (Program.receiveThread != null && Program.receiveThread.IsAlive)

{

Program.receiveThread.Abort();

}

Program.networkStream?.Close();

Program.tcpClient?.Close();

}

private void Form2\_FormClosed(object sender, EventArgs e)

{

this.Close();

}

}

}

## Приложение A.4.

## Файл « Form2.cs» для Клиента

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace ClientApp

{

public struct UserMessage

{

public string Login { get; set; }

public string Message { get; set; }

}

public partial class Form2 : Form

{

readonly Form1 form1Instace = Form1.Instance;

public List<UserMessage> messages = new List<UserMessage>();

public string generalChat = "";

public string notificationText = "";

public Form2()

{

InitializeComponent();

this.Text = "Чат";

listUsers.Items.Add("Общий чат");

listUsers.SelectedIndex = 0;

loginLabel.Text = form1Instace.login;

this.FormClosing += Form2Closing;

Program.SendMessage("getusers/");

messageTextBox.KeyPress += MessageBox\_KeyPress;

}

private void MessageBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == (char)Keys.Enter)

{

SendBtn\_Click(sender, e);

}

}

public event EventHandler Form2Closed;

private void SendBtn\_Click(object sender, EventArgs e)

{

string message = messageTextBox.Text;

if (!string.IsNullOrWhiteSpace(message))

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

Program.SendMessage("message\_for/" + selectedUser + "/" + form1Instace.login + "/(" + DateTime.Now.ToString("HH:mm") + ") " + form1Instace.login + ": " + message);

messageTextBox.Text = string.Empty;

}

else

{

Program.SendMessage("message/(" + DateTime.Now.ToString("HH:mm") + ") " + form1Instace.login + ": " + message);

messageTextBox.Text = string.Empty;

}

}

}

public void UpdateMessage(string login, string message)

{

for (int i = 0; i < messages.Count; i++)

{

if (messages[i].Login == login)

{

var tempUserMessage = messages[i];

tempUserMessage.Message = messages[i].Message + (message + Environment.NewLine);

messages[i] = tempUserMessage;

if (listUsers.InvokeRequired)

{

listUsers.Invoke((MethodInvoker)delegate

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

if (login != selectedUser)

{

notificationText += message + Environment.NewLine;

DisplayMessage(notificationTextBox, notificationText);

}

}

else

{

notificationText += message + Environment.NewLine;

DisplayMessage(notificationTextBox, notificationText);

}

});

}

else

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

if (login != selectedUser)

{

notificationText += message + Environment.NewLine;

DisplayMessage(notificationTextBox, notificationText);

}

}

else

{

notificationText += message + Environment.NewLine;

DisplayMessage(notificationTextBox, notificationText);

}

}

break;

}

}

}

public void UpdateMessages()

{

if (listUsers.InvokeRequired)

{

listUsers.Invoke((MethodInvoker)delegate

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

foreach (UserMessage message in messages)

{

if (message.Login == selectedUser)

{

DisplayMessage(messagesTextBox, message.Message);

}

}

}

else

{

DisplayMessage(messagesTextBox, generalChat);

}

});

}

else

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

foreach (UserMessage message in messages)

{

if (message.Login == selectedUser)

{

DisplayMessage(messagesTextBox, message.Message);

}

}

}

else

{

DisplayMessage(messagesTextBox, generalChat);

}

}

}

public void DisplayMessage(TextBox textBox, string message)

{

if(textBox.InvokeRequired)

{

textBox.Invoke((MethodInvoker)delegate

{

textBox.Text = message;

});

}

else

{

textBox.Text = message;

}

}

public void DisplayUser(string login)

{

if (listUsers.InvokeRequired)

{

listUsers.Invoke((MethodInvoker)delegate

{

listUsers.Items.Add(login);

});

}

else

{

listUsers.Items.Add(login);

}

}

public void RemoveUser(string login)

{

if (listUsers.InvokeRequired)

{

listUsers.Invoke((MethodInvoker)delegate

{

string selectedUser = listUsers.SelectedItem.ToString();

if (selectedUser == login)

{

listUsers.SelectedIndex = 0;

}

listUsers.Items.Remove(login);

});

}

else

{

string selectedUser = listUsers.SelectedItem.ToString();

if (selectedUser == login)

{

listUsers.SelectedIndex = 0;

}

listUsers.Items.Remove(login);

}

for (int i = 0; i < messages.Count; i++)

{

if (messages[i].Login == login)

{

messages.RemoveAt(i);

break;

}

}

}

private void ListUsers\_SelectedIndexChanged(object sender, EventArgs e)

{

UpdateMessages();

}

private void Form2Closing(object sender, FormClosingEventArgs e)

{

Form2Closed?.Invoke(this, EventArgs.Empty);

}

private void ClearBtn\_Click(object sender, EventArgs e)

{

notificationText = "";

DisplayMessage(notificationTextBox, notificationText);

}

}

}