Министерство науки и высшего образования Российской Федерации Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2023

Выполнил:

студент группы 21ВВП1

Тюрин В.И.

Принял:

Егоров В.Ю.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Основы операционных систем»

на тему «Система программ обмена текстовыми сообщениями»

# Содержание

[Содержание 4](#_Toc155694143)

[Введение 5](#_Toc155694144)

[1. Постановка задачи 6](#_Toc155694145)

[2. Теоретическая часть задания 7](#_Toc155694146)

[2.1. Общие теоретические сведения 7](#_Toc155694147)

[2.2. Протокол TCP 9](#_Toc155694148)

[3. Описание алгоритма работы 13](#_Toc155694149)

[4.1. Клиент 15](#_Toc155694150)

[4.2. Сервер 16](#_Toc155694151)

[5. Описание работы программы 18](#_Toc155694152)

[Заключение 21](#_Toc155694153)

[Список используемых источников 22](#_Toc155694154)

[Приложение A. Листинг программы 23](#_Toc155694155)

[Приложение A.1. Файл «Server.cpp» для Сервера 23](#_Toc155694156)

[Приложение A.2. Файл «Program.cs» для Клиента 26](#_Toc155694157)

[Приложение A.3. Файл «Form1.cs» для Клиента 29](#_Toc155694158)

[Приложение A.4. Файл « Form2.cs» для Клиента 31](#_Toc155694159)

# 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.

# 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

## Общие теоретические сведения

Схема клиент-сервер представляет схему организации вычислительных или сетевых систем, где задачи и сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и их потребителями, которые называются клиентами. Фактически клиент и сервер представляют программное обеспечение, обычно размещенное на различных вычислительных устройствах, и взаимодействующее друг с другом через вычислительные сети, используя сетевые протоколы. Однако они также могут быть размещены на одной машине. При поступлении нескольких запросов одновременно, они ставятся 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 основном 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, используя протоколы.

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. 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.

## Протокол TCP

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым (в отличие от UDP) целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализации TCP обычно встроены в ядра ОС. Существуют реализации TCP, работающие в пространстве пользователя.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надёжную передачу потока байтов от одного процесса к другому. TCP реализует управление потоком, управление перегрузкой, рукопожатие, надёжную передачу.

Каждый сегмент на нижних уровнях TCP/IP обрабатывается индивидуально. То есть, как минимум, он будет запакован в индивидуальный пакет. Пакеты идут по сети и промежуточные маршрутизаторы в общем случае уже ничего не знают о том, что запаковано в эти пакеты. Часто пакеты с целью балансировки нагрузки могут идти по сети разными путями, через разные промежуточные устройства, с разной скоростью. Таким образом получатель, декапсулировав их, может получить сегменты не в том порядке, в котором они отправлялись. TCP автоматически пересоберёт их в нужном порядке используя всё то же поле порядковых номеров и передаст после склейки на уровень приложений.

Перед началом передачи полезных данных TCP позволяет убедиться в том, что получатель существует, слушает нужный отправителю порт и готов принимать данные для этого устанавливается сессия при помощи механизма трёхстороннего рукопожатия (three-way handshake). Сервер должен отправить клиенту подтверждение (ACK) вместе с порядковым номером (SYN) сервера. В свою очередь клиент отвечает АСК, и соединение устанавливается. Далее, в рамках сессии передаются полезные пользовательские данные. После завершения передачи сессия закрывается, тем самым получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён.

Порт источника, порт назначения - это 16-битные поля, которые содержат номера портов — числа, которые определяются по специальному списку. Порт источника идентифицирует приложение клиента, с которого отправлены пакеты. Ответные данные передаются клиенту на основании этого номера. Порт назначения идентифицирует порт, на который отправлен пакет.

Порядковый номер выполняет две задачи: если установлен флаг SYN, то это изначальный порядковый номер — ISN (Initial Sequence Number), и первый байт данных, которые будут переданы в следующем пакете, будет иметь номер, равный ISN + 1. В противном случае, если SYN не установлен, первый байт данных, передаваемый в данном пакете, имеет этот порядковый номер. Поскольку поток TCP в общем случае может быть длиннее, чем число различных состояний этого поля, то все операции с порядковым номером должны выполняться по модулю 232. Это накладывает практическое ограничение на использование TCP. Если скорость передачи коммуникационной системы такова, чтобы в течение MSL (максимального времени жизни сегмента) произошло переполнение порядкового номера, то в сети может появиться два сегмента с одинаковым номером, относящихся к разным частям потока, и приёмник получит некорректные данные.

Acknowledgment Number (ACK SN) (32 бита) - если установлен бит ACK, то это поле содержит порядковый номер октета, который отправитель данного сегмента желает получить. Это означает, что все предыдущие октеты (с номерами от ISN+1 до ACK-1 включительно) были успешно получены. Длина заголовка - это поле, которое определяет размер заголовка пакета TCP в 4-байтных (4-октетных) словах. Минимальный размер составляет 5 слов, а максимальный — 15, что составляет 20 и 60 байт соответственно. Смещение считается от начала заголовка TCP. Зарезервировано (6 бит) для будущего использования и должно устанавливаться в ноль. Из них два (5-й и 6-й) уже определены:

CWR (Congestion Window Reduced) — Поле «Окно перегрузки уменьшено» — флаг установлен отправителем, чтобы указать, что получен пакет с установленным флагом ECE (RFC 3168) ECE (ECN-Echo) — Поле «Эхо ECN» — указывает, что данный узел способен на ECN (явное уведомление перегрузки) и для указания отправителю о перегрузках в сети (RFC 3168) Флаги (управляющие биты) — это поле, которое содержит 6 битовых флагов: URG — поле «Указатель важности» задействовано (англ. Urgent pointer field is significant) ACK — поле «Номер подтверждения» задействовано (англ. Acknowledgement field is significant) PSH — (англ. Push function) инструктирует получателя протолкнуть данные, накопившиеся в приёмном буфере, в приложение пользователя RST — оборвать соединения, сбросить буфер (очистка буфера) (англ. Reset the connection) SYN — синхронизация номеров последовательности (англ. Synchronize sequence numbers). FIN (англ. final, бит) — флаг, будучи установлен, указывает на завершение соединения (англ. FIN bit used for connection termination). Размер окна - количество байт данных начиная с последнего номера подтверждения, которые может принять отправитель данного пакета. Иначе говоря, отправитель пакета располагает для приема данных буфером длиной "размер окна" байт.

Поле контрольной суммы — это 16-битное дополнение к сумме всех 16-битных слов заголовка (включая псевдозаголовок) и данных. Если сегмент, по которому вычисляется контрольная сумма, имеет длину не кратную 16-ти битам, то длина сегмента увеличивается до кратной 16-ти, за счет дополнения к нему справа нулевых битов заполнения. Биты заполнения (0) не передаются в сообщении и служат только для расчёта контрольной суммы. При расчёте контрольной суммы значение самого поля контрольной суммы принимается равным 0.

Указатель важности - 16-битовое значение положительного смещения от порядкового номера в данном сегменте. Это поле указывает порядковый номер октета, которым заканчиваются важные данные. Поле принимается во внимание только для пакетов с установленным флагом URG. Используется для внеполосных данных. Опции могут применяться в некоторых случаях для расширения протокола. Иногда используются для тестирования. На данный момент в опции практически всегда включают 2 байта NOP (в данном случае 0x01) и 10 байт, задающих timestamps. Вычислить длину поля опции можно через значение поля смещения. Обмен данными, ориентированный на соединения, может использовать надежную связь, для обеспечения которой протокол уровня 4 посылает подтверждения о получении данных и запрашивает повторную передачу, если данные не получены или искажены.

# 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.



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

Изображение выглядит как текст, диаграмма, круг, линия

Автоматически созданное описание

Рисунок 2 - UML-диаграмма вариантов использования.

1. **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 текст, чек, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3 - Общая схема работы клиента.

## Сервер

Программа сервера использует библиотеку Winsock2 для работы с сетевыми соединениями. Вот пошаговое описание организации работы сервера:

Инициализация Winsock2: Программа начинается с инициализации библиотеки Winsock2 с помощью функции WSAStartup.

Создание сокета сервера: Сервер создает сокет с помощью функции socket, указывая параметры семейства протоколов (AF\_INET для IPv4), типа сокета (SOCK\_STREAM для TCP) и протокола (0 для автоматического выбора).

Привязка сокета к адресу и порту: Устанавливается адрес и порт для сокета сервера с помощью структуры sockaddr\_in, после чего выполняется вызов bind для привязки сокета к указанному адресу и порту.

Ожидание подключения клиентов: Вызов listen позволяет сокету начать ожидание подключений. Сервер ждет, пока клиент подключится с помощью функции accept. Когда клиент подключается, создается новый сокет для общения с этим клиентом.

Создание нити для обработки клиента: При подключении клиента создается новая нить выполнения (std::thread), которая вызывает функцию HandleClient, передавая ей информацию о клиенте (структуру User). В этой нити происходит обработка запросов от клиента.

Обработка запросов в HandleClient: Функция HandleClient находится в бесконечном цикле и ожидает получение данных от клиента с помощью функции recv. Затем она интерпретирует эти данные, выполняет соответствующие действия, например, отправляет сообщение всем клиентам или обновляет список подключенных пользователей.

Хранение данных о пользователе: Данные о каждом подключенном пользователе (логин и сокет) хранятся в векторе std::vector<User\*> users.

Закрытие сокета при отключении пользователя: При отключении пользователя его сокет закрывается с помощью функции closesocket, а затем данные пользователя удаляются из вектора users.

Отправка данных клиентам: Сервер отправляет данные клиентам с помощью функции send. Например, рассылка сообщения всем подключенным клиентам, кроме отправителя.

Изображение выглядит как текст, снимок экрана, Шрифт, чек

Автоматически созданное описание

Рисунок 4 - Общая схема работы сервера.

# Описание работы программы

Результаты работы программы изображены на рисунках 5-11.

Сначала нужно запустить сервер (Рисунок 5).

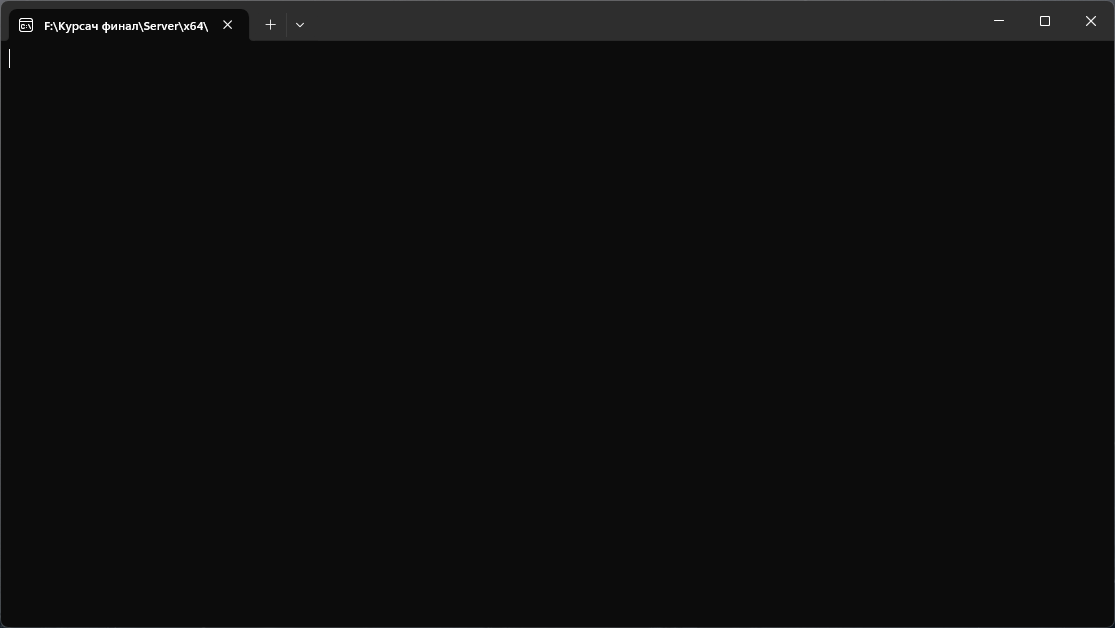


Рисунок 5 - Окно серверного приложения.

Далее необходимо запустить клиентское приложение (Рисунок 7) и ввести имя, а за затем нажать кнопку «Log in». 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).

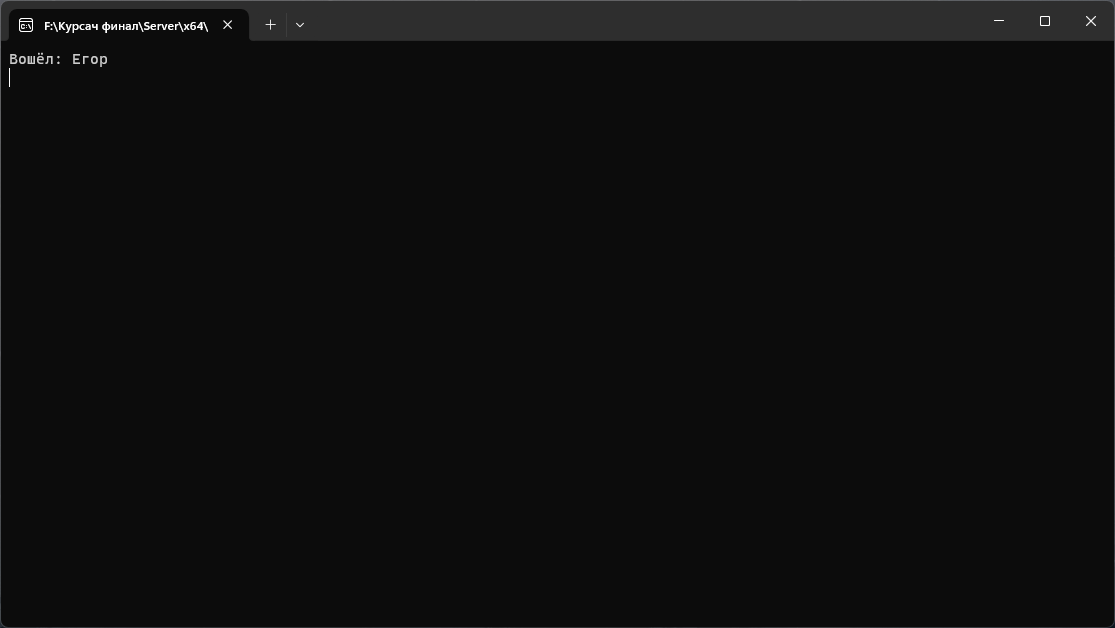


Рисунок 6 - Окно серверного приложения.

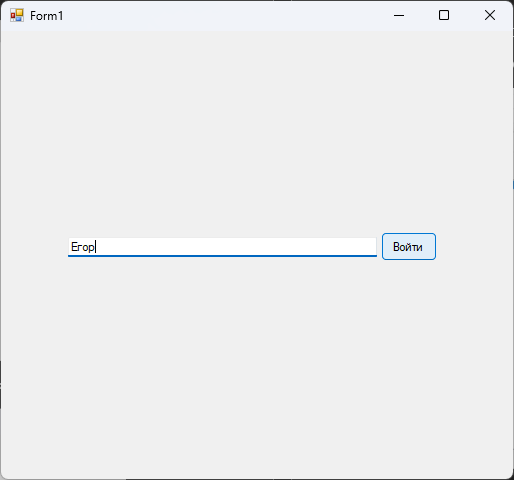
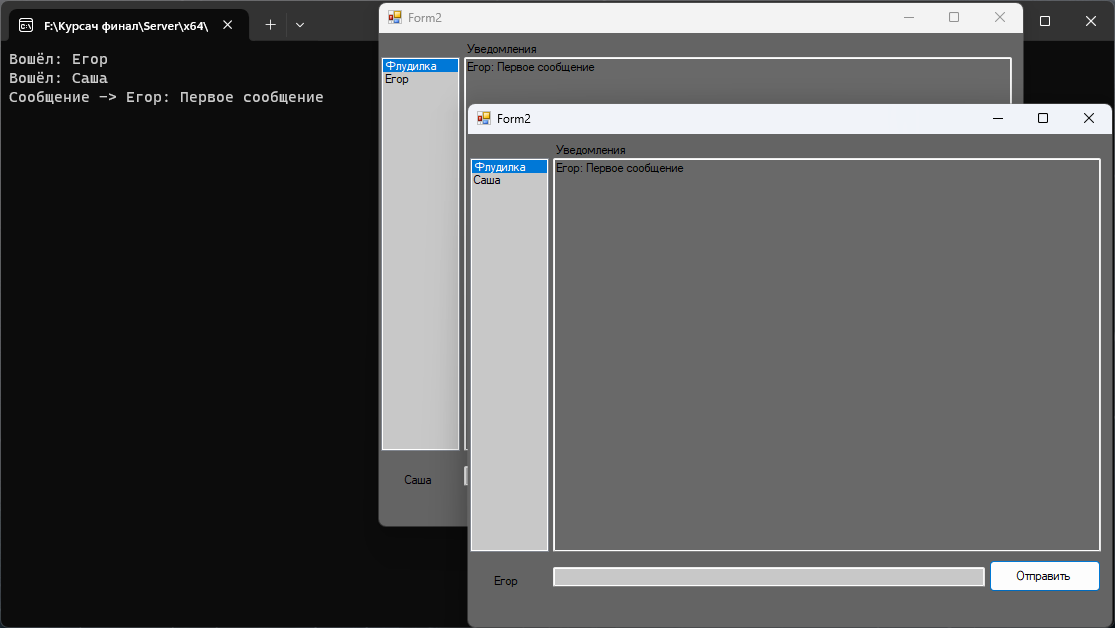
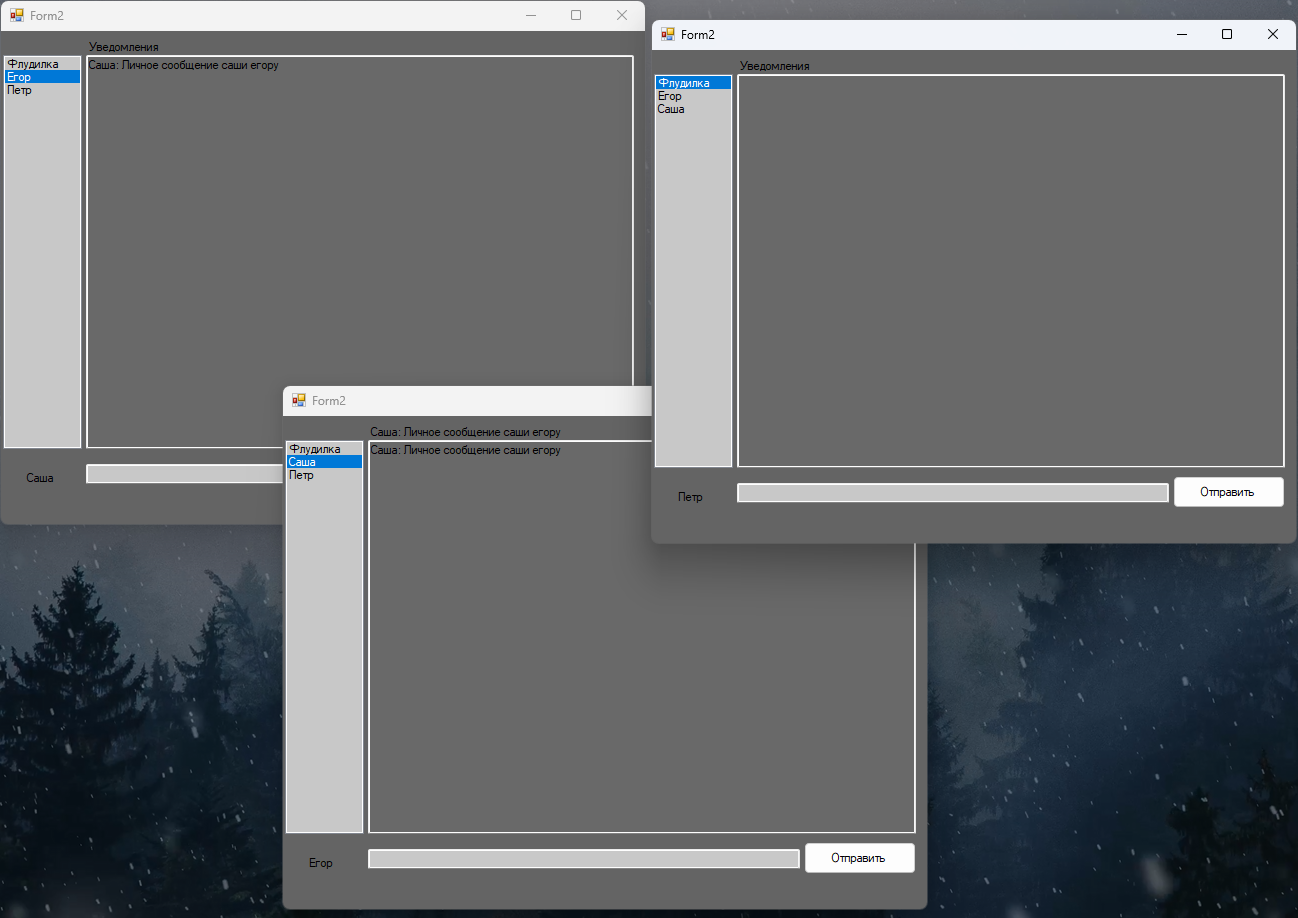


Рисунок 7 - Окно клиентского приложения.

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



**Рисунок 8 – Окно клиентского приложения с отправленным сообщением и окно сервера.**



**Рисунок 10 – Окно клиентского 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>

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

struct User {

SOCKET clientSocket;

char login[32] = "";

};

std::vector<User\*> users;

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 << "Сообщение -> " << message << std::endl;

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

for (const User\* otherUser : users) {

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

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

}

}

}

else if (command.find("messageto|") == 0) {

const char\* prefix = "messageto|";

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);

for (const User\* otherUser : users) {

if (otherUser->login == login) {

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

}

}

sprintf\_s(buffer, "%s%s", "messagefr|", request);

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

}

}

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 << "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) {

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) {

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 << "Отключился: " << user->login << std::endl;

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() {

SetConsoleOutputCP(CP\_UTF8);

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(5400);

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

listen(serverSocket, 5);

while (true) {

User\* newUser = new User;

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

users.push\_back(newUser);

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

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

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

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net.Sockets;

using System.Runtime.Remoting.Messaging;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

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", 5400);

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.UTF8.GetString(buffer, 0, bytesRead);

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

{

string message = receivedMessage.Substring(8);

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

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

form1Instace.form2.UpdateMessages();

}

}

else if (receivedMessage.StartsWith("messageto|"))

{

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("messagefr|"))

{

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;

if (form1Instace != null)

{

form1Instace.form2.RemoveUser(login);

}

}

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

{

MessageBox.Show($"Login already exists", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

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

{

Form1 form1Instace = Form1.Instance;

if (form1Instace != null)

{

form1Instace.createForm2();

}

}

}

}

public static void SendMessage(string message)

{

try

{

byte[] buffer = Encoding.UTF8.GetBytes(message);

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

}

catch (Exception ex)

{

MessageBox.Show($"Error sending message: {ex.Message}", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

## Приложение A.3. Файл «Form1.cs» для Клиента

using System;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.Http;

using System.Net.Sockets;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ClientApp

{

public partial class Form1 : Form

{

private static Form1 instance;

public Form1()

{

InitializeComponent();

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

instance = this;

this.FormClosing += Form1Closing;

}

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("Please enter both login.", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (login.Contains("|"))

{

MessageBox.Show("The login contains a forbidden character '|'.", "Warning", 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();

}

if (Program.networkStream != null)

{

Program.networkStream.Close();

}

if (Program.tcpClient != null)

{

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.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ClientApp

{

public struct UserMessage

{

public string Login { get; set; }

public string Message { get; set; }

}

public partial class Form2 : Form

{

Form1 form1Instace = Form1.Instance;

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

public string generalChat = "";

public Form2()

{

InitializeComponent();

listUsers.Items.Add("Флудилка");

listUsers.SelectedIndex = 0;

loginLabel.Text = form1Instace.login;

this.FormClosing += Form2Closing;

Program.SendMessage("getusers|");

}

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("messageto|" + selectedUser + "|" + form1Instace.login + "|" + form1Instace.login + ": " + message);

messageTextBox.Text = string.Empty;

}

else

{

Program.SendMessage("message|" + 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)

{

labelNotif.Text = message;

}

}

else

{

labelNotif.Text = message;

}

});

}

else

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

if (login != selectedUser)

{

labelNotif.Text = message;

}

}

else

{

labelNotif.Text = message;

}

}

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(message.Message);

}

}

}

else

{

DisplayMessage(generalChat);

}

});

}

else

{

if (listUsers.SelectedIndex != 0)

{

string selectedUser = listUsers.SelectedItem.ToString();

foreach (UserMessage message in messages)

{

if (message.Login == selectedUser)

{

DisplayMessage(message.Message);

}

}

}

else

{

DisplayMessage(generalChat);

}

}

}

public void DisplayMessage(string message)

{

if(messagesTextBox.InvokeRequired)

{

messagesTextBox.Invoke((MethodInvoker)delegate

{

messagesTextBox.Text = message;

});

}

else

{

messagesTextBox.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 labelNotif\_Click(object sender, EventArgs e)

{

}

private void messagesTextBox\_TextChanged(object sender, EventArgs e)

{

}

}

}