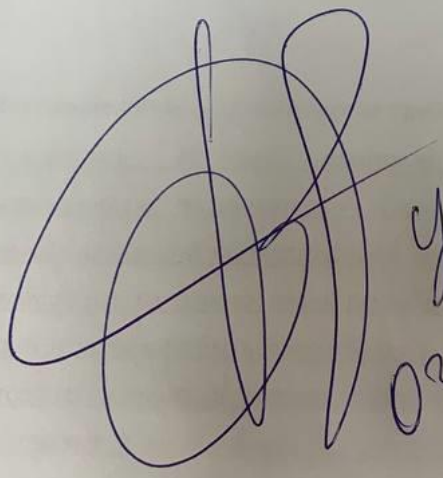
Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Основы операционных систем»
на тему «Система программ обмена текстовыми сообщениями»



Выполнил студент
группы 21ВВП2:
Рузляев Д. А.

Принял:
к.т.н., доцент
Егоров В.Ю.

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ
М. А. Митрохин
«5» октября 2023 г.

ЗАДАНИЕ

на курсовое проектирование по дисциплине

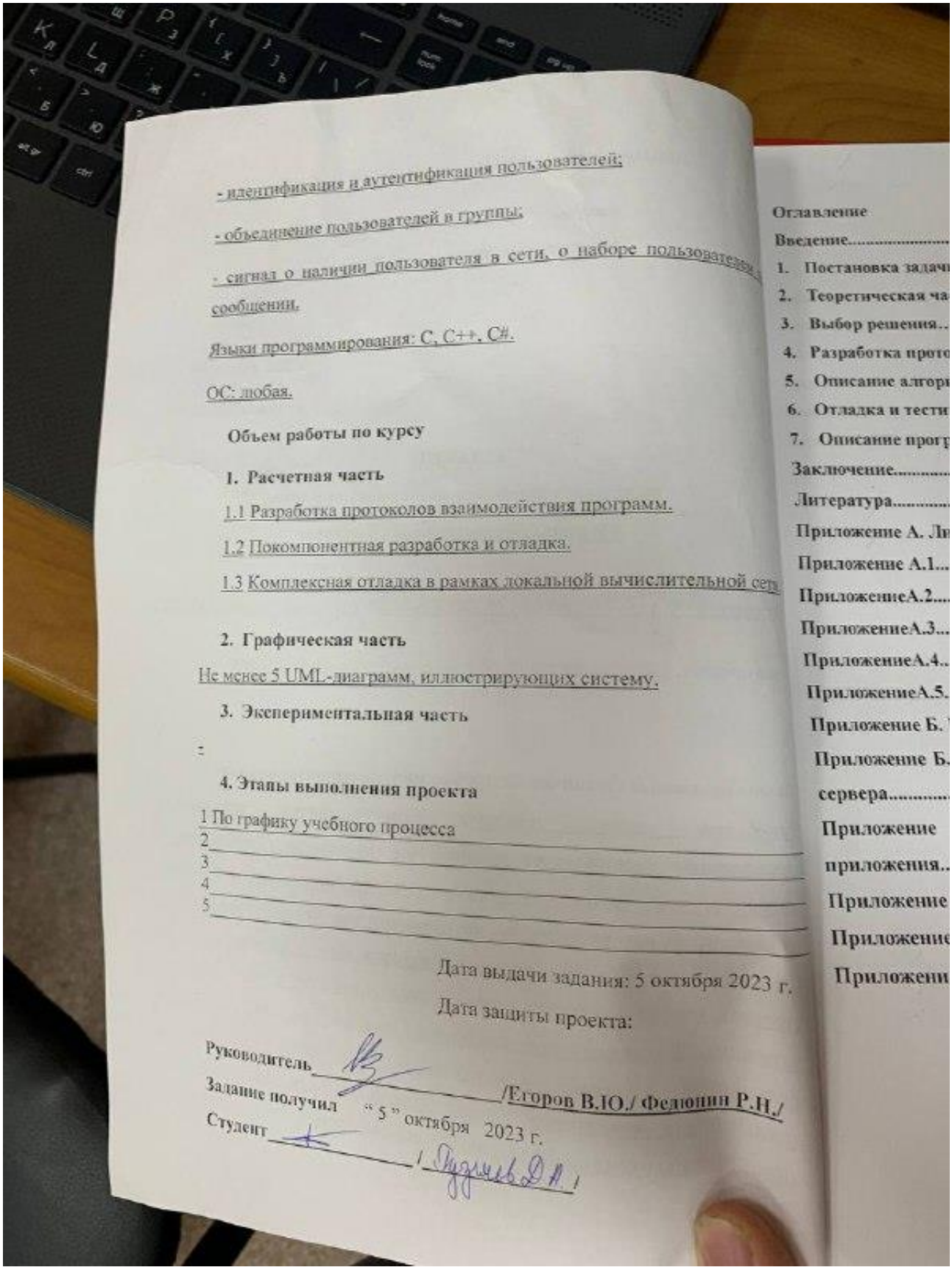
Основы операционных систем

Студенту Гудкову Дмитрию Александровичу Группа 21ВВП2

Тема проекта: Система программ обмена текстовыми сообщениями

Исходные данные (технические требования) на проектирование:

Разработать систему программ клиент-серверной архитектуры, позволяющую пользователям обмениваться текстовыми сообщениями в рамках локальной вычислительной сети. Взаимодействие клиента и сервера происходит на основе вновь разрабатываемого протокола взаимодействия. Протокол может быть представлен в текстовом виде / бинарном виде. На сторонах клиента и сервера протокол реализовывается в виде конечных автоматов. Программа-клиент реализуется графическим интерфейсом для пользователя, а сервер – консольными интерфейсами. Необходимо предусмотреть возможность обмена индивидуальными сообщениями и широковещательными сообщениями. Опционально реализуется функционал:



- идентификация и аутентификация пользователей;

- объединение пользователей в группы;

- сигнал о наличии пользователя в сети, о наборе пользователем сообщения.

Языки программирования: C, C++, C#.

ОС: любая.

Объем работы по курсу

1. Расчетная часть

1.1 Разработка протоколов взаимодействия программ.

1.2 Покомпонентная разработка и отладка.

1.3 Комплексная отладка в рамках локальной вычислительной сети

2. Графическая часть

Не менее 5 UML-диаграмм, иллюстрирующих систему.

3. Экспериментальная часть

:

4. Этапы выполнения проекта

1 По графику учебного процесса
2
3
4
5

Дата выдачи задания: 5 октября 2023 г.

Дата защиты проекта:

Руководитель _____ /Егоров В.Ю./ Федюнин Р.Н./

Задание получил " 5 " октября 2023 г.

Студент _____ / Гудиев Д.А. /

Оглавление

Введение.....
1. **Постановка задачи**
2. **Теоретическая ча**
3. **Выбор решения..**
4. **Разработка прото**
5. **Описание алгори**
6. **Отладка и тести**
7. **Описание прогр**

Заключение.....
Литература.....
Приложение А. Ли
Приложение А.1...
ПриложениеА.2....
ПриложениеА.3...
ПриложениеА.4..
ПриложениеА.5.
Приложение Б.
Приложение Б.
сервера.....
Приложение
приложения..
Приложение
Приложение
Приложени

Оглавление

Введение 6

1. Постановка задачи 7

2. Теоретическая часть задания 8

3.Выбор решения 10

4.Разработка протокола взаимодействия программ 11

5. Описание алгоритма работы 12

6. Отладка и тестирование 13

7.Описание работы программы 14

Заключение 18

Литература 19

Приложение А. Листинг программы 20

Приложение А.1.Файл «ConnectedClient.cs» Server. 20

ПриложениеА.2. Файл «Program.cs» Server. 21

ПриложениеА.3. Файл «MainViewModel.cs» Client. 23

ПриложениеА.4. Файл «App.xaml.cs» Client. 25

ПриложениеА.5. Файл «MainWindow.xaml» Client. 26

ПриложениеБ. UML-диаграммы 28

Приложение Б.1.UML – диаграмма протокола взаимодействия клиента и сервера 28

Приложение Б.2.UML – диаграмма вариантов использования приложения……...29

Приложение Б.3. UML – диаграмма взаимодействия пользователей. 30

Приложение Б.4.UML – диаграмма состояний клиента. 31

Приложение Б.5. UML – диаграмма состояний сервера. 32

Введение

«Клиент - сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

1. Постановка задачи

Необходимо разработать систему программ клиент-серверной архитектуры, позволяющую пользователям осуществлять обмен текстовыми сообщениями между пользователями в рамках локальной вычислительной сети. Взаимодействие клиента и сервера происходит на основе вновь разрабатываемого протокола взаимодействия. Протокол может быть представлен в текстовом виде / бинарном виде. На сторонах клиента и сервера протокол реализовывается в виде конечных автоматов. Программа клиент реализуется графическим интерфейсом для пользователя, а сервер – консольными интерфейсами. Необходимо предусмотреть возможность обмена индивидуальными сообщениями и широковещательными сообщениями. Опционально реализуется функционал:

- идентификация и аутентификация пользователей;
- объединение пользователей в группы;
- сигнал о наличии пользователя в сети, о наборе пользователем текста в сообщении.

Языки программирования: C, C++, C#.

ОС: любая.

2. Теоретическая часть задания

В сети Internet объединено множество компьютеров различных типов. Эти компьютеры могут использовать разные операционные системы, но все они должны поддерживать принятый для обмена данными в Internet стандарт, реализованный на базе стека протоколов TCP/IP (Transmission Control Protocol / Internet Protocol).

Протоколы транспортного уровня широко применяются в современных сетях. Именно они позволяют гарантировать доставку сообщения до адресата, а также сохраняют правильную последовательность передачи данных. При этом протоколы имеют ряд различий, что позволяет использовать их профильно, для решения своих задач каждый.

Протокол TCP– это сетевой протокол, который «заточен» под соединение. Иными словами, прежде чем начать обмен данными, данному протоколу требуется установить соединение между двумя хостами. Данный протокол имеет высокую надежность, поскольку позволяет не терять данные при передаче, запрашивает подтверждения о получении от принимающей стороны и в случае необходимости отправляет данные повторно. При этом отправляемые пакеты данных сохраняют порядок отправки, то есть можно сказать, что передача данных упорядочена.

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 надежную передачу потока байтов от одной программы на некотором компьютере к

другой программе на другом компьютере (например, программы для электронной почты, для обмена файлами). TCP контролируетдлинусообщения, скоростьобменасообщениями, сетевойтрафик.

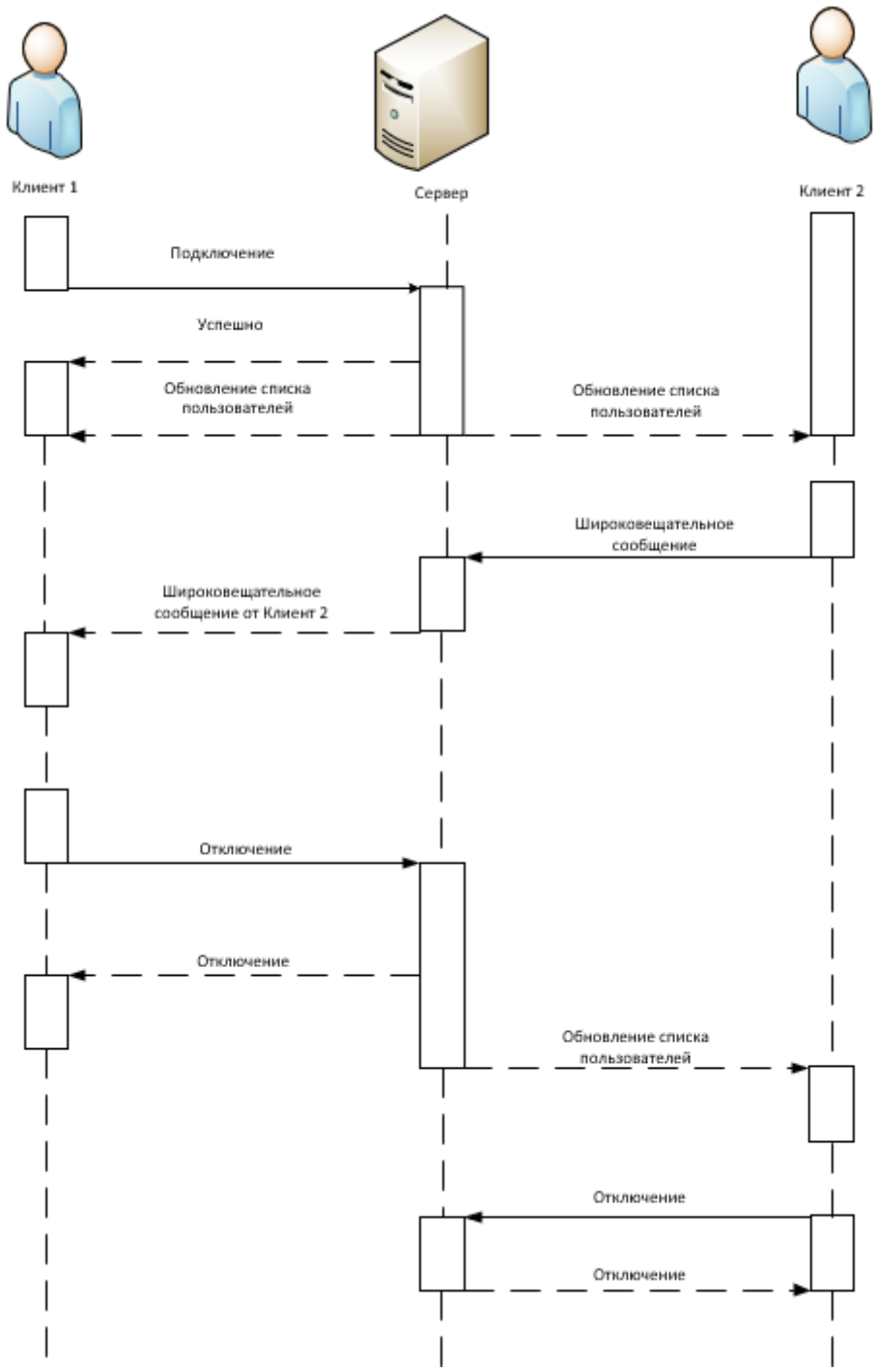


Рисунок 1. Схема работы протокола TCP.

3.Выбор решения

Приложение реализовано на языке C#.

C# (произносится си шарп) — объектно-ориентированный язык программирования общего назначения. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Delphi, Модула, Smalltalk и, в особенности, Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

4.Разработка протокола взаимодействия программ

Сразу после включения сервера, открывается ранее заданный порт, а затем создаёт коллекцию “ConnectedClients” (собственный кастомный класс в котором объединен “TcpClient” и “Nick”).

Затем запускается слушание и наступает этап ожидания подключения клиента. После подключения клиента, все значения перемещаются на новый поток, в котором происходит обработка клиента, а затем происходит ожидание нового клиента.

При подключении клиента, происходит чтение его строки никнейма, если же такой никнейм уже зарегистрирован то подключится не получится.

Если всё успешно, то происходит чтение текстовой строки и её «рассылка» остальным пользователям.

5. Описание алгоритма работы

Разработанная программа основывается на вышеописанном протоколе и состоит из клиента и сервера.

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 нити в функцию обработки клиентов. 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.

6. Отладка и тестирование

Отладка и тестирование проводились средствами MicrosoftVisualStudio 2022. Во время отладки и тестирования не было выявлено ошибок программы.

7.Описание работы программы

Ниже представлена работа клиент-серверного приложения.

Для начала необходимо запустить сервер:

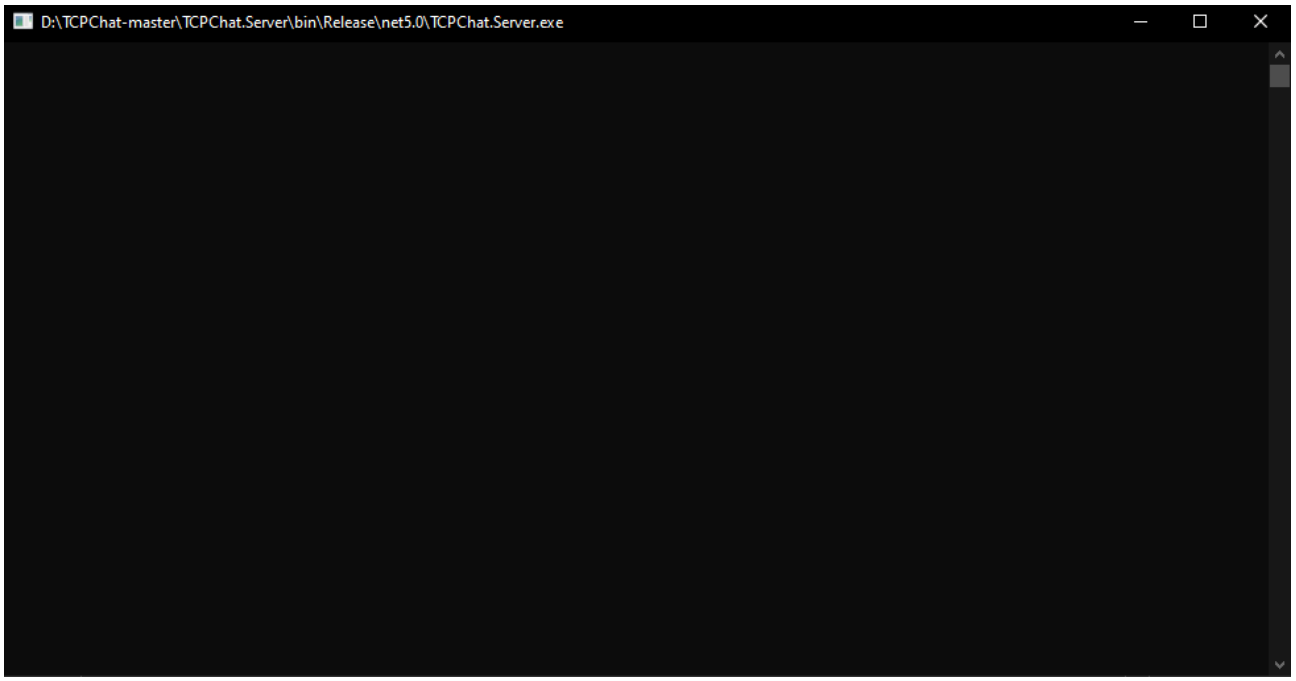


Рисунок 2. Окно серверного приложения.

Далее запускаем экземпляр клиента:

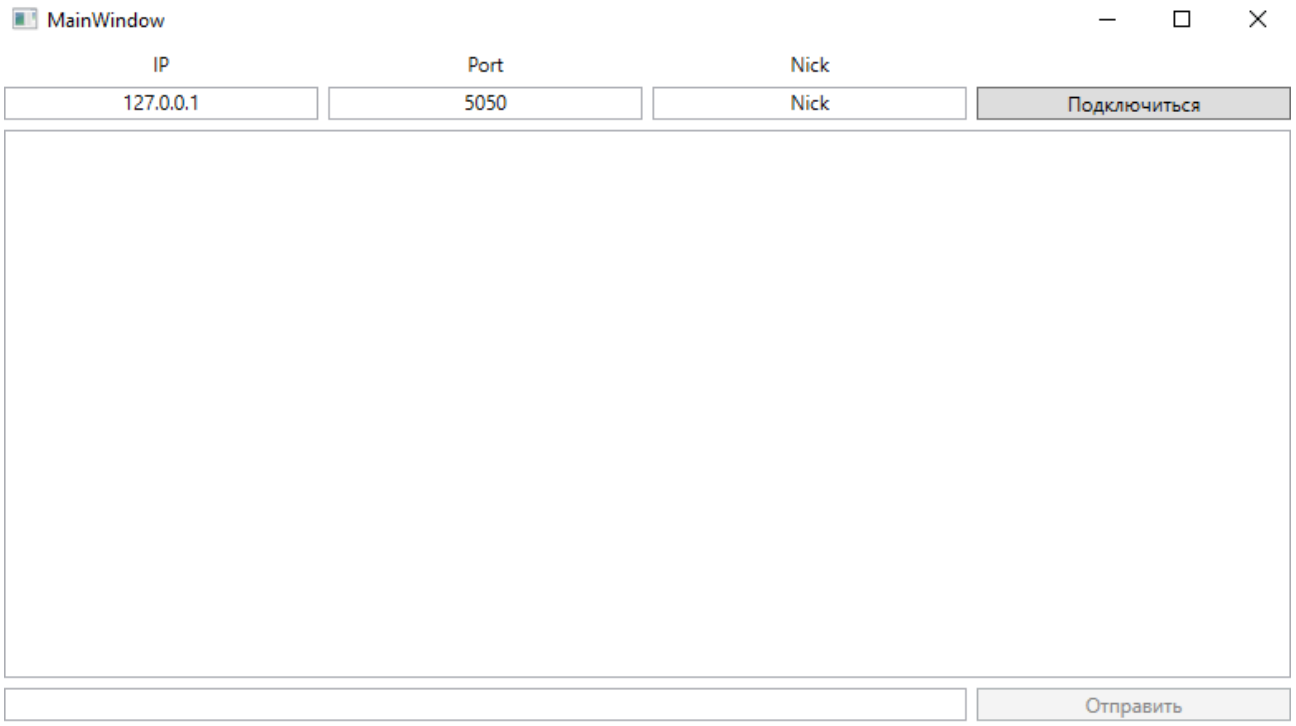


Рисунок 3.Окно клиентского приложения.

Вводим имя клиента, ip и порт сервера. Нажимаем кнопку подключиться.

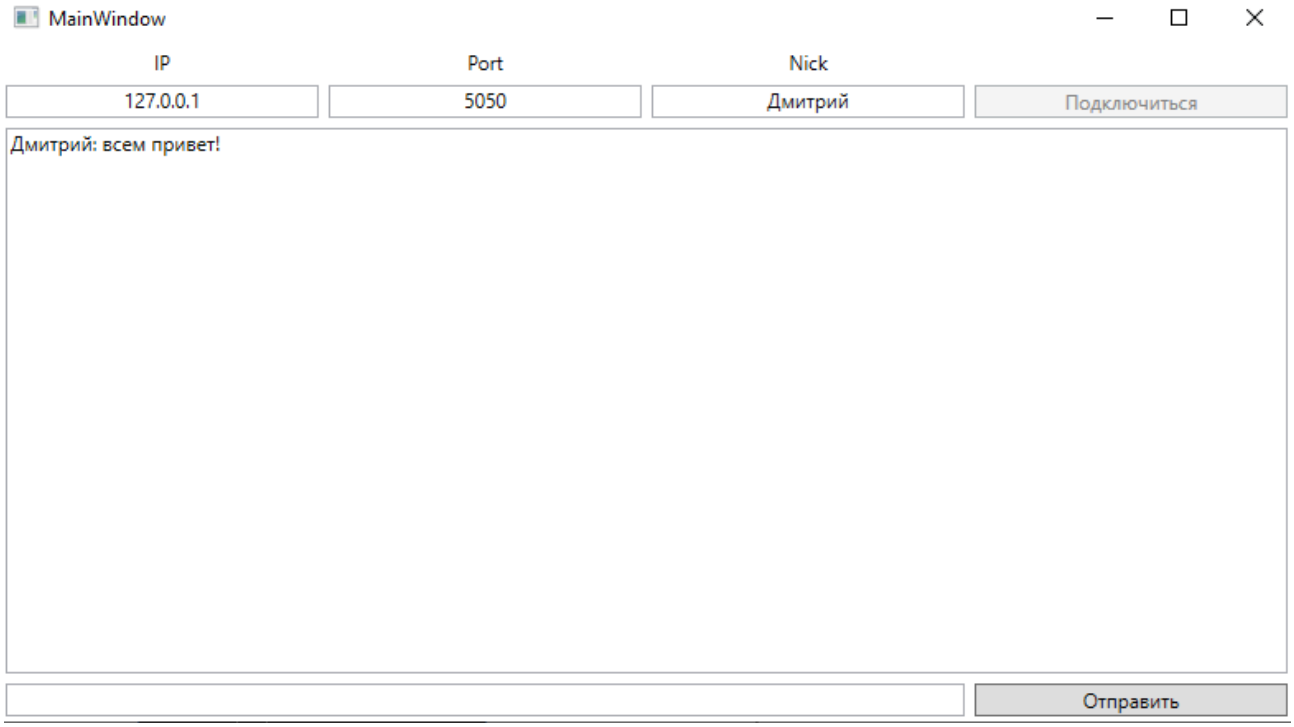


Рисунок 4. Подключение клиента к серверу.

На сервереотображается подключение:

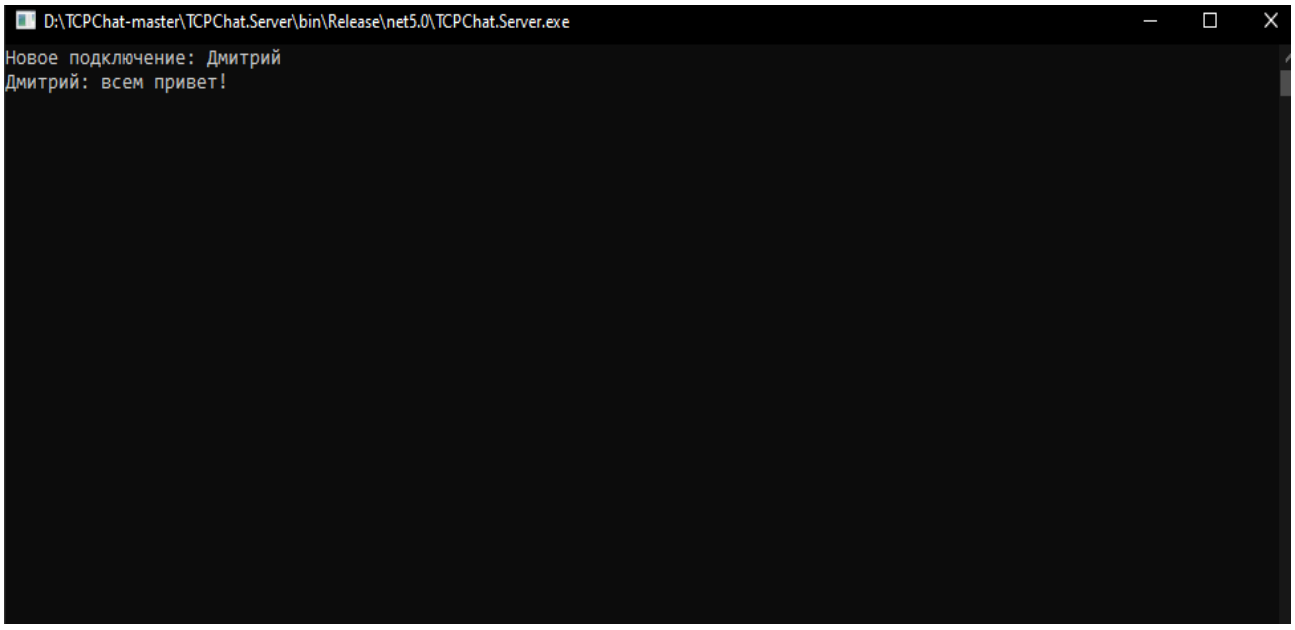


Рисунок 5. Подключение клиента на сервере.

Далее запускаем и подключаем к серверу еще пару экземпляров клиента:

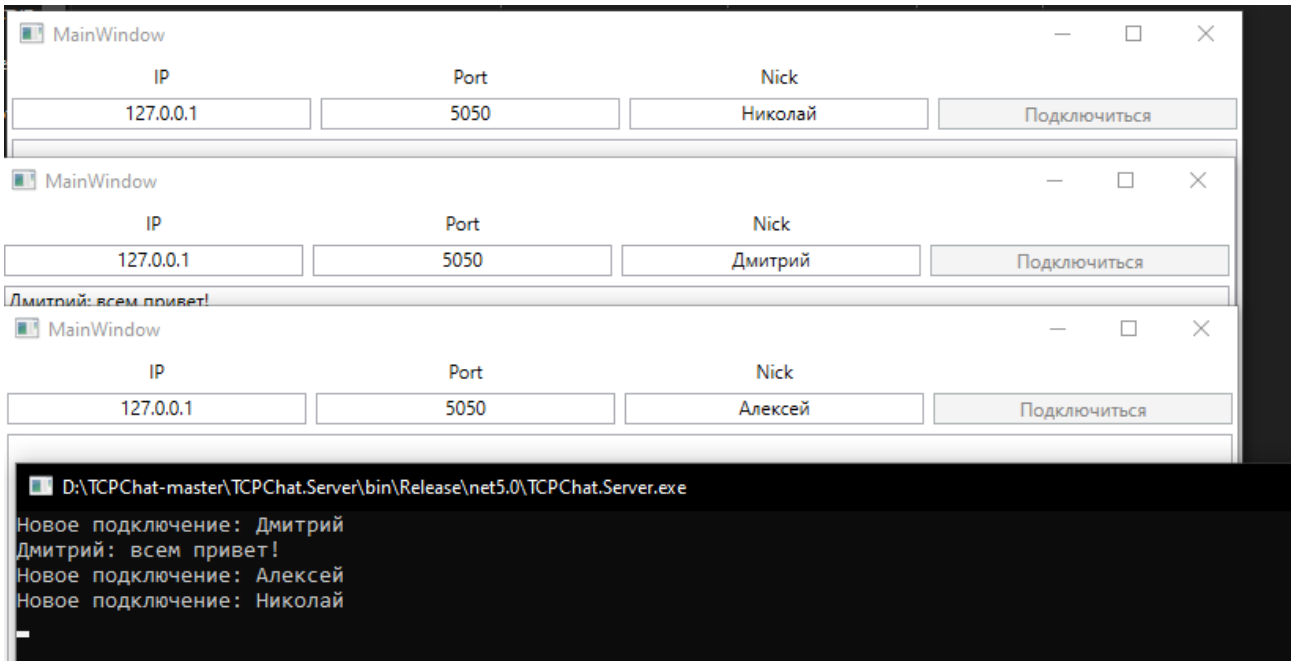


Рисунок 6. Несколько экземпляров клиента.

Чтобы отправить сообщение всем пользователям, необходимо в поле ввода набрать текст и нажать кнопку «отправить»:

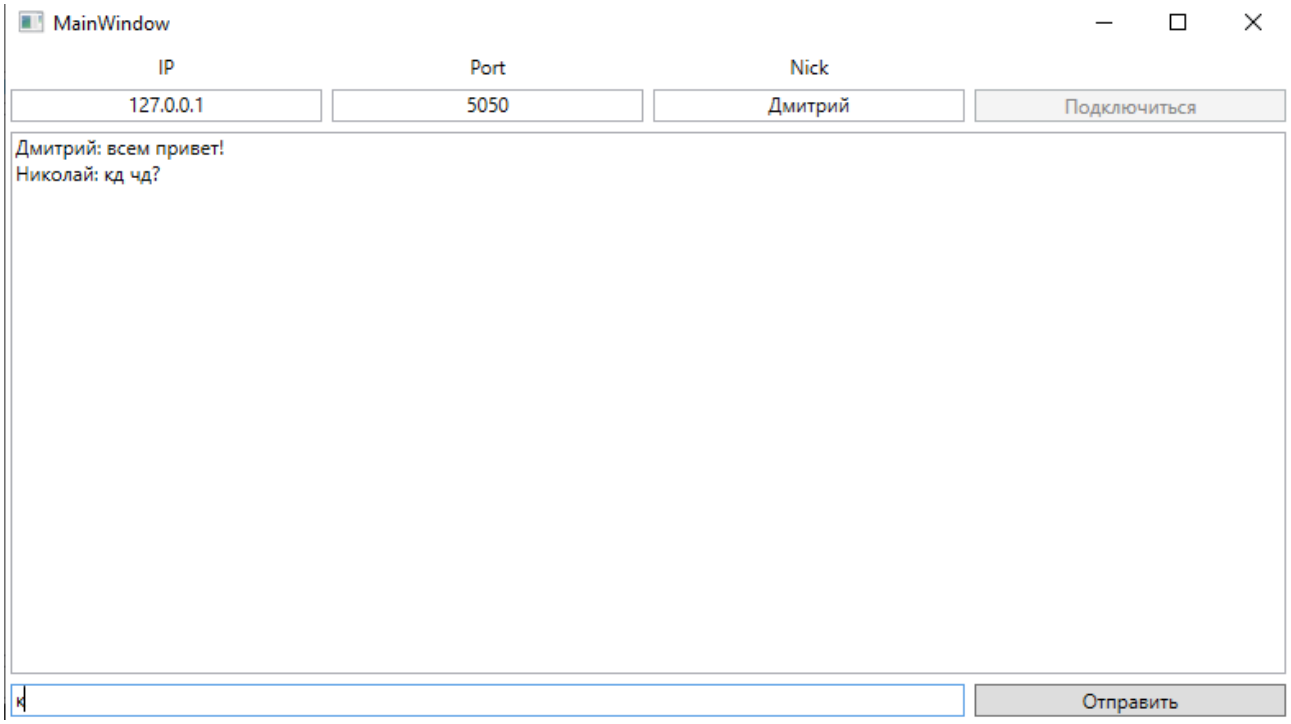


Рисунок 7. Пример отправки сообщениявсем пользователям в клиентском приложении.

Чтобы покинуть чат, необходимо закрыть окно:

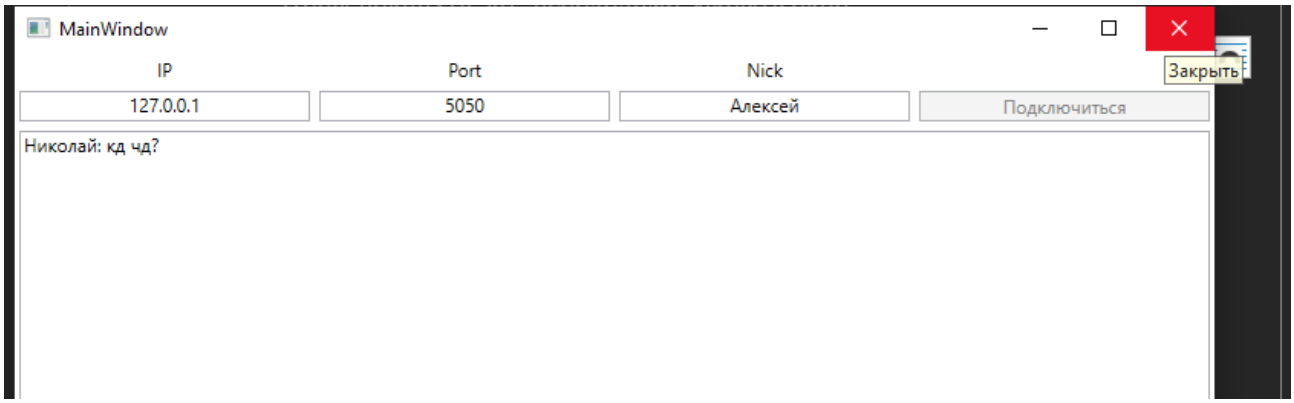


Рисунок 8. Отключение пользователя в клиентском приложении.

Заключение

В ходе выполнения курсового проектирования были изучены принципы работы протоколов TCP/IP. Были получены навыки разработки объектно-ориентированного приложения с пользовательским интерфейсом.

В итоге было разработано клиент-серверное приложение для передачи текстовых сообщений с использованием протокола TCP/IP.

Литература

1. Рихтер Джеффри. – Windows для профессионалов.Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. 2001 г.
2. Джепикс Филипп, Троелсен Эндрю – Язык программирования C# 7 и платформы .NET и .NET Core. 2018
3. Шилд Г. – Полный справочник по C#. : Пер. с англ. — М. : Издательский дом "Вильяме", 2004.

Приложение А. Листинг программы

Приложение А.1.Файл «ConnectedClient.cs» Server.

```
using System.Net.Sockets;

namespace TCPChat.Server
{
    public class ConnectedClient
    {
        public TcpClient Client { get; set; } //объединениетцп
        public string Name { get; set; } // и имени

        public ConnectedClient(TcpClient client, string name)
        {
            Client = client;
            Name = name;
        }
    }
}
```

ПриложениеА.2. Файл «Program.cs» Server.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Threading.Tasks;

namespace TCPChat.Server
{
   internal class Program
   {
      static TcpListener listener = new TcpListener(IPAddress.Any, 5050);//
прилюбоминтернетподключенииоткрываетпорт
      static List<ConnectedClient> clients = new List<ConnectedClient>();// кастомныйкласс
      static void Main(string[] args)
{
listener.Start();

while (true)// после подключения клиента переносим его на новый поток, обрабатываем,ждем
следующего
{
            var client = listener.AcceptTcpClient();

Task.Factory.StartNew(() =>
            {
               var sr = new StreamReader(client.GetStream());// авторизация
               while (client.Connected)
               {
                  varline = sr.ReadLine();// клиент подключился, читаем строку логина
var nick = line.Replace("Login: ", "");

                  if (line.Contains("Login: ") && !string.IsNullOrWhiteSpace(nick))
                  {
                     if (clients.FirstOrDefault(s =>s.Name == nick) is null)
                     {
clients.Add(new ConnectedClient(client, nick));
Console.WriteLine($"Новоеподключение: {nick}");
                        break;
                     }
                     else
                     {
                        var sw = new StreamWriter(client.GetStream());// проверяемнадубликат
sw.AutoFlush = true;
sw.WriteLine("Пользователь с таким ником уже есть");
client.Client.Disconnect(false);
                     }
                  }
               }

               while(client.Connected)
               {
                  try
                  {
                     var line = sr.ReadLine();

SendToAllClients(line);

Console.WriteLine(line);
                  }
                  catch (Exception ex)
```

```
                {
                    Console.WriteLine(ex.Message);//строка сообщения, отправляем
}
                }
            });
        }
    }

    private static void SendToAllClients(string message)
    {
Task.Run(() =>
        {
for(int i = 0; i<clients.Count; i++)
            {
                try
                {
                    if (clients[i].Client.Connected)
                    {
                        var sw = new StreamWriter(clients[i].Client.GetStream());
sw.AutoFlush = true;
sw.WriteLine(message);
                    }
                    else
                    {
Console.WriteLine($"{clients[i].Name} отключился");//удаляемклиентакоторыйнеподключён
clients.RemoveAt(i);
                    }
                }
                catch (Exception ex)
                {
Console.WriteLine(ex.Message);
                }
            }
        });
    }
  }
}
```

ПриложениеА.3. Файл «MainViewModel.cs» Client.

```
using DevExpress.Mvvm;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace TCPChat.Client
{
   public class MainViewModel :ViewModelBase
   {
      public string IP { get; set; } = "127.0.0.1";
      public int Port { get; set; } = 5050;
      public string Nick { get; set; } = "Nick";
      public string Chat
      {
         get =>GetValue<string>();
         set =>SetValue(value);
      }
      public string Message { get =>GetValue<string>(); set =>SetValue(value); }

      private TcpClient? _client;
      privateStreamReader? _reader; //предназначен для ввода символов в определенной кодировке Stream
, а класс предназначен для ввода и вывода байтов. Используется StreamReader для чтения строк данных из
стандартного текстового файла.
      privateStreamWriter? _writer; //предназначен для вывода символов в определенной кодировке, а классы,
производные от Stream , предназначены для байтового ввода и вывода.

publicMainViewModel()
      {

      }
// циклкоторыйхочетчитатьstream
private void Listener()
      {
Task.Run(() =>
         {
            while (true)
            {
               try
               {
                  if (_client?.Connected == true)
                  {
                     var line = _reader?.ReadLine();
                     if (line != null)
{                     // работа с текстом
Chat += line + "\n";
}
                     else
                     {
                        _client.Close();
                        Chat += "Connected error.\n";
                     }
                  }
Task.Delay(10).Wait(); //экономияресурсовцп
```

```
                }
                catch (Exception ex)
                {
                    Chat += ex.Message + "\n";
                }
            }
        });
    }
    public AsyncCommandConnectCommand
    {
        get
        {
            return new AsyncCommand(() =>
            {
                return Task.Run(() =>
                {
                    try
                    {
                        _client = new TcpClient();
                        _client.Connect(IP, Port);
                        _reader = new StreamReader(_client.GetStream());//
                        _writer = new StreamWriter(_client.GetStream());//
Listener();
                        _writer.AutoFlush = true;

                        _writer.WriteLine($"Login: {Nick}");
                    }
                    catch (Exception ex)
                    {
MessageBox.Show(ex.Message);
                    }
                });
            }, () => _client is null || _client?.Connected == false);
        }
    }

    public AsyncCommandSendCommand
    {
        get
        {
            return new AsyncCommand(() =>
            {
                return Task.Run(() =>
                {
                    _writer?.WriteLine($"{Nick}: {Message}");
                    Message = "";
                });
            }, () => _client?.Connected == true, !string.IsNullOrWhiteSpace(Message));
        }
    }
  }
}
```

ПриложениеА.4. Файл «App.xaml.cs» Client.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace TCPChat.Client
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
    }
}
```

ПриложениеА.5. Файл «MainWindow.xaml» Client.

```
<Window x:Class="TCPChat.Client.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:TCPChat.Client"
mc:Ignorable="d"
      Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:MainViewModel/>
</Window.DataContext>
<Window.Resources>
<Style TargetType="TextBox">
<Setter Property="Margin" Value="3"/>
<Setter Property="TextAlignment" Value="Center"/>
</Style>
<Style TargetType="TextBlock">
<Setter Property="Margin" Value="3"/>
<Setter Property="TextAlignment" Value="Center"/>
</Style>
<Style TargetType="Button">
<Setter Property="Margin" Value="3"/>
</Style>
</Window.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<StackPanel>
<UniformGrid Columns="4">
<TextBlock Text="IP"/>
<TextBlock Text="Port"/>
<TextBlock Text="Nick"/>
</UniformGrid>
<UniformGrid Columns="4">
<TextBox Text="{Binding IP}"/>
<TextBox Text="{Binding Port}"/>
<TextBox Text="{Binding Nick}"/>
<Button Command="{Binding ConnectCommand}" Content="Подключиться"/>
```

```
</UniformGrid>
</StackPanel>

<TextBoxGrid.Row="1"
            Text="{Binding Chat}"
TextAlignment="Left"
IsReadOnly="True"
IsReadOnlyCaretVisible="True"/>

<Grid Grid.Row="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="3*"/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<TextBoxMaxLines="5"
              Text="{Binding Message}"
TextAlignment="Left"
TextWrapping="Wrap"/>
<Button
Grid.Column="1"
          Command="{Binding SendCommand}"
          Content="Отправить"/>
</Grid>
</Grid>
</Window>
```

ПриложениеБ. UML-диаграммы

Приложение Б.1.UML – диаграмма протокола взаимодействия клиента и сервера

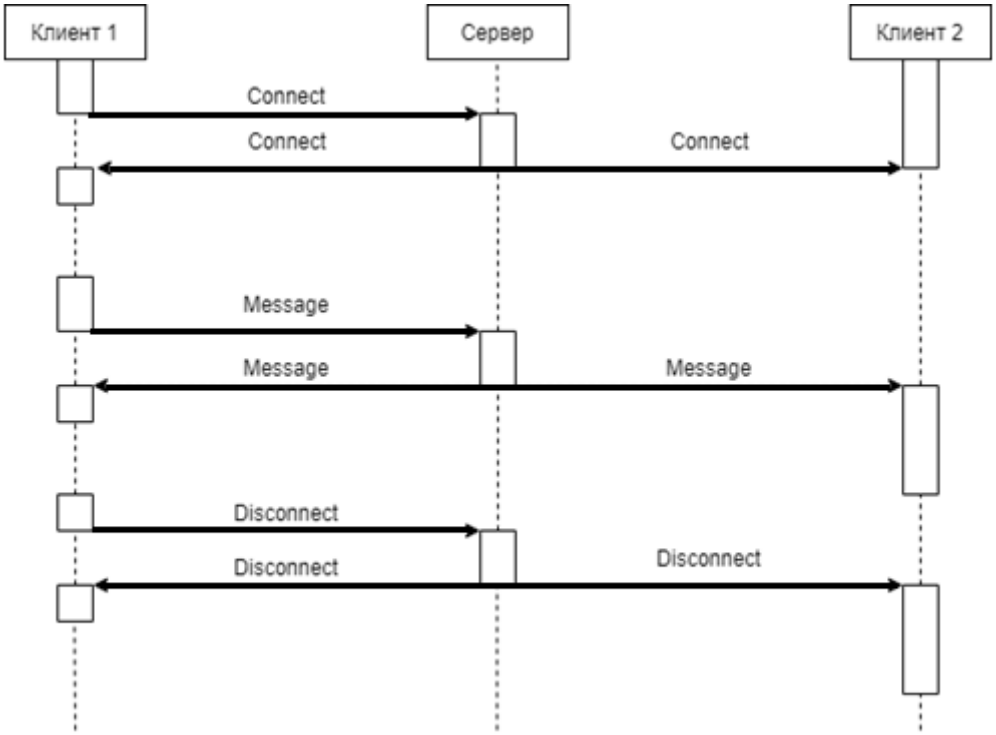


Рисунок 9.UML - диаграмма протокола взаимодействия клиента и сервера.

Приложение Б.2.UML – диаграмма вариантов использования приложения.

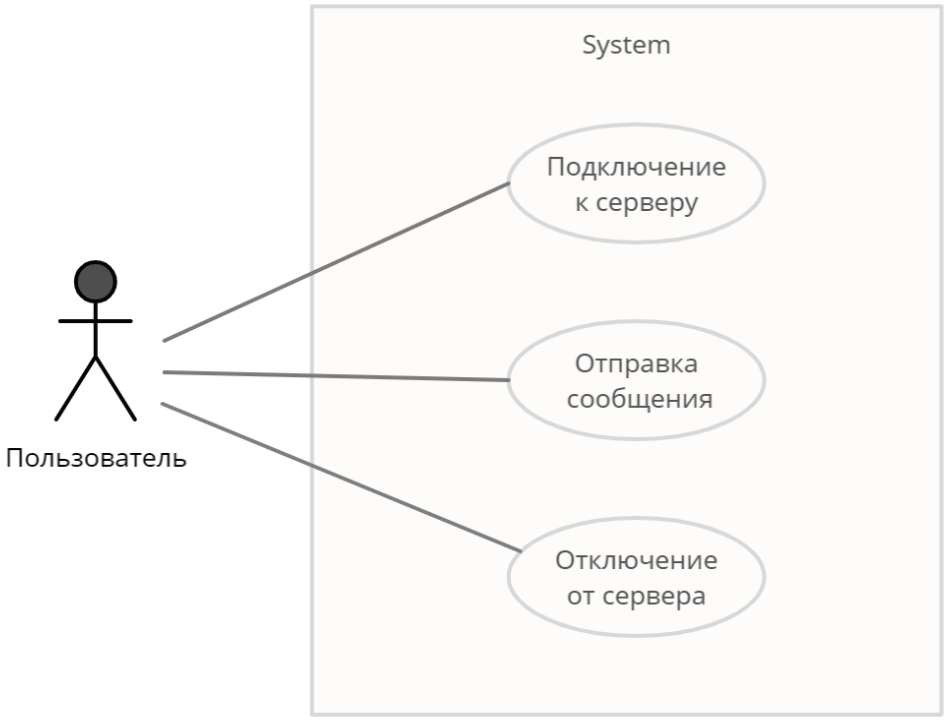


Рисунок 10. UML - диаграмма вариантов использования приложения.

Приложение Б.3. UML – диаграмма взаимодействия пользователей.

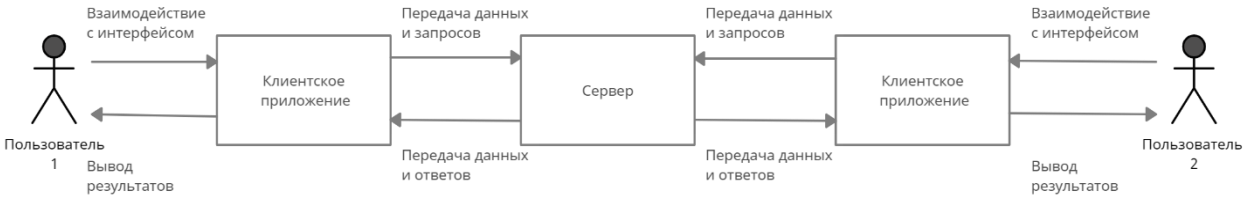


Рисунок 11.UML - диаграмма взаимодействия пользователей.

Приложение Б.4.UML – диаграмма состояний клиента.

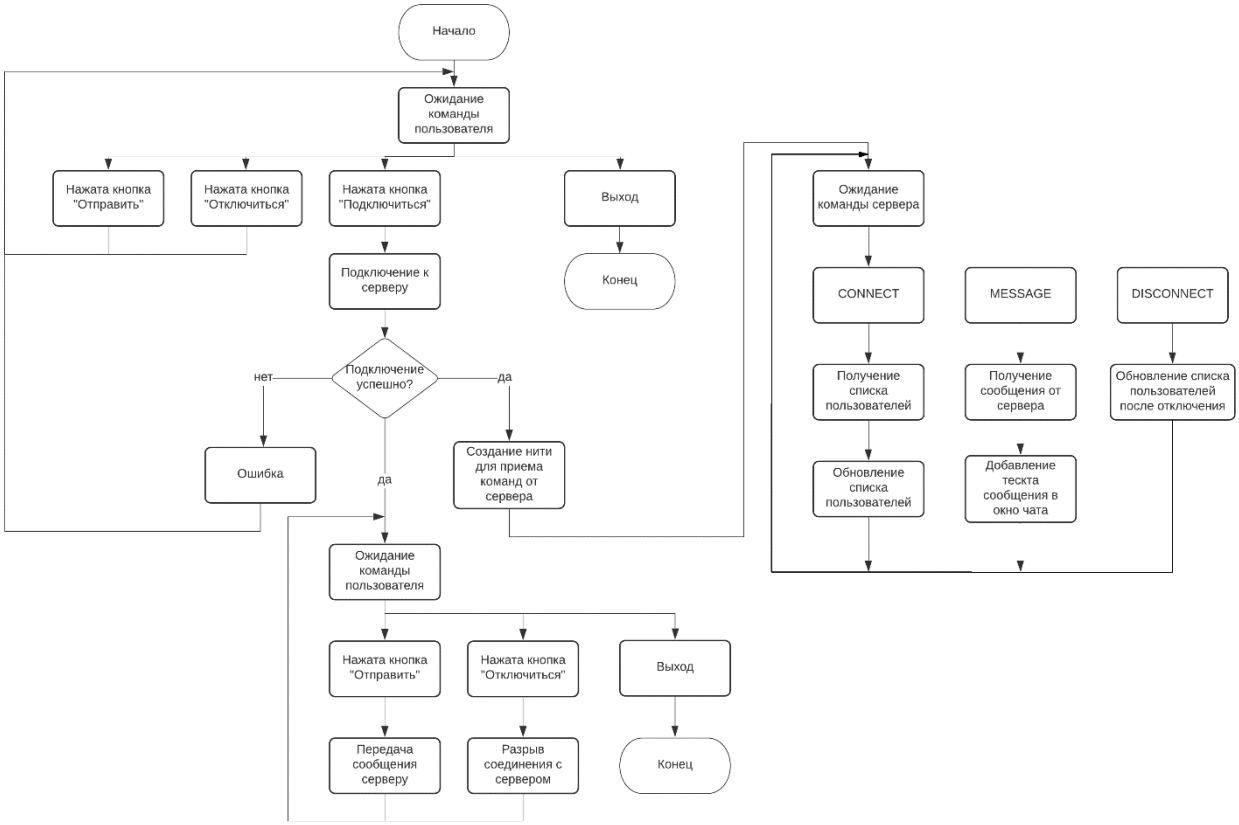


Рисунок 12. UML - диаграмма состояний клиента.

Приложение Б.5. UML – диаграмма состояний сервера.

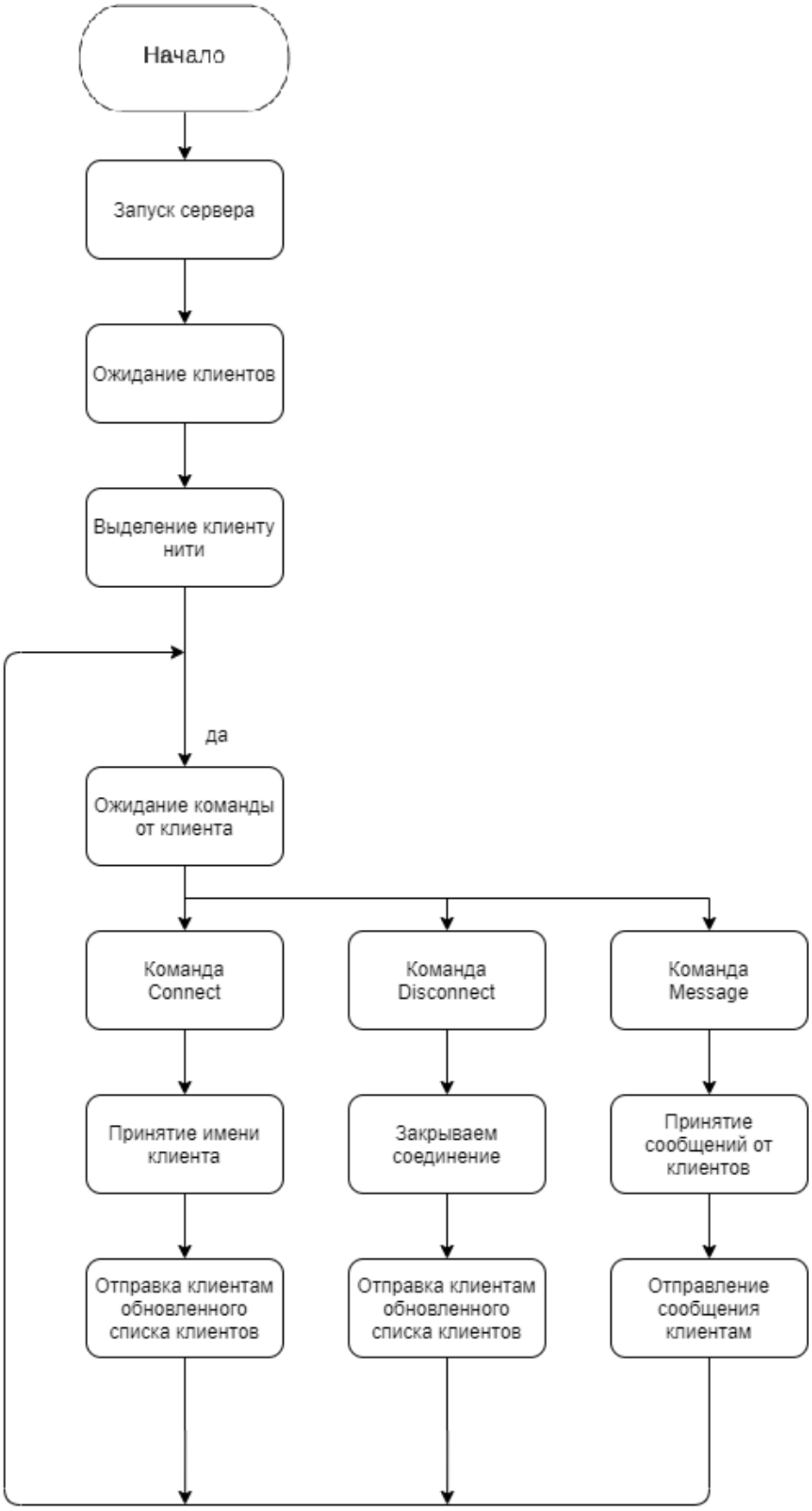


Рисунок 13. UML - диаграмма состояний сервера.