

Министерство образования Российской Федерации

Пензенский Государственный университет

Кафедра “Вычислительная техника”

Пояснительная записка

К курсовому проектированию

По курсу «Технологии программирования»

На тему: «Система программ обмена текстовыми сообщениями»

Выполнил студент группы 19ВВ1:

Гусев В.

Принял:

Дорошенко И.Н.

Пенза 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М. А. Митрохин

«__»__ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Техническое программирование
Студенту Будель Виталий Акимович Группа 9БВ1
Тема проекта Система передачи данных по сети

Исходные данные (технические требования) на проектирование

Разработать систему передачи данных - серверной архитектуры, позволяющую осуществлять передачу текстовых сообщений между пользователями.
Функции сервера:
идентификация пользователя;
прием и передача информации;
сбор статистики о пользователе;
Функции клиента:
визуальный интерфейс пользователя;
сетевое взаимодействие с сервером;
В рамках курсовой работы необходимо разработать программную систему, позволяющую реализовать заданные требования:
Операционные системы: Windows, Linux
Языки программирования: C, C++, C#, Java

Объем работы по курсу

1. Расчетная часть

1. Разработка проекта взаимодействия процессов
2. Разработка программы

2. Графическая часть

1. Разработка не менее 5 схем-диаграмм, иллюстрирующих логическое взаимодействие функциональных систем

3. Экспериментальная часть

1. Отладка программы

Срок выполнения проекта по разделам

1. Согласно графика выполнения работы

2
3
4
5
6
7
8

Дата выдачи задания "21" 01.2022

Дата защиты проекта "18" 03.2022

Руководитель Доросенко

Задание получил "21" января 2022 г.

Студент Гусев В.А.

Оглавление

Введение	5
1. Постановка задачи.....	6
2. Теоретическая часть	7
3. Моделирование и реализация ПО	10
4. Заключение	23
5. Список используемых источников	24
6. Приложение	25
Приложение А	25
Приложение В	30
Результат работы программы	30

Введение

Архитектура «Клиент-Сервер» (также используются термины «сеть Клиент-Сервер» или «модель Клиент-Сервер») предусматривает разделение процессов предоставления услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняет свои задачи независимо от других.

В архитектуре «Клиент-Сервер» несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – сервера (server – англ. «официант, услуга»), которая также может называться хост-системой (host system, от host – англ. «хозяин», обычно гостиницы).

Клиентская машина предоставляет пользователю т.н. «дружественный интерфейс» (user-friendly interface), чтобы облегчить его взаимодействие с сервером.

В качестве среды разработки мною была выбрана Microsoft VisualStudio2019, язык программирования – C#.

Целью данной курсовой работы является разработка программы на языке C#, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется приложение клиент-серверной архитектуры, которое будет осуществлять передачу текстовых сообщений.

1.Постановка задачи

Требуется реализовать программу, которая осуществит передачу текстового сообщения согласно клиент-серверной архитектуре.

Программа должна содержать графический интерфейс, корректно осуществлять передачу текстовых сообщений, правильно закрываться в случае экстренного выхода из системы.

Устройство ввода - клавиатура и мышь.

2. Теоретическая часть

Технология «Клиент – сервер» - представляет собой архитектуру программного комплекса, которая распределяет прикладную программу на два логически различных компонента, а именно клиент и сервер, которые взаимодействуют по схеме «запрос-ответ» решая свои определенные задачи (рисунок 1).



Рисунок 1 – Архитектура «Клиент - Сервер»

Компьютер (или программа), которая управляет или владеет каким-либо ресурсом, называется сервером этого ресурса.

Компьютер (или программа), который запрашивает и пользуется каким-либо ресурсом, называется клиентом этого ресурса.

Клиенты и серверы могут располагаться как на одном персональном компьютере (ПК), так и на разных ПК объединенные сетью. Также могут возникать такие ситуации, когда некоторые программные блоки одновременно выполняют функции сервера, работая с одним блоком и работу клиента работая с другим.

Двухзвенная архитектура «Клиент-сервер»

Классическая архитектура клиент-сервер распределяет три основные части приложения на два физических модуля. Обычно прикладной компонент располагается на сервере (например, сервере базы данных), компонент представления данных - на стороне клиента, а компонент управления ресурсом распределяется между клиентской и серверной

частями. В этом заключается основной недостаток классической двухуровневой архитектуры.

Используя двухзвенную архитектуру во время распределения алгоритма обрабатывания данных, разработчик должен обладать полной информацией о том, какие последние изменения, были внесены в систему, и какова значимость этих изменений, из-за чего возникает большая сложность во время разработки клиент-серверной системы, её установки и обслуживания. Так как такие мероприятия стоят значительных усилий, и постоянной координации действий совершаемые разными группами специалистов.

Чтобы предотвратить сопутствующую несогласованность работы различных элементов такой архитектуры специалисты создали две модификации для двухзвенной архитектуры «Клиент – сервер»: одна модификация стала называться «Толстый клиент» («Тонкий сервер»), а другая «Тонкий клиент» («Толстый сервер»).

В данной двухзвенной архитектуре разработчики пытались перенести процесс выполнения обработки данных на одну из двух физических частей - на сторону клиента («Толстого клиента»), или на сторону сервера («Тонкого клиента»).

Каждый из описанных подходов обладал своими недостатками. В первом подходе неоправданно перегружалась сеть, так как по ней передавались необработанные а, следовательно, избыточного размера данные. Также был усложнён процесс поддержки системы и процесс её изменения, так как, чтобы заменить алгоритм для вычисления или исправления ошибок нужно одновременно полностью заменить весь интерфейс в программах, чтобы не возникли ошибки или несогласованности в обработке данных. Если же весь процесс обработки информации

выполнялся бы на сервере, то возникла бы проблема в процессе описания встроенных процедур и процесса их отладки. Систему, обрабатывающую информацию на сервере невозможно было ни как перенести на другую платформу или операционную систему, без серьезных сбоев в работе.

В том случае если всё равно будет требоваться разработка двухуровневой классической архитектуры «Клиент – сервер», то надо знать следующее: структура архитектуры «Толстый сервер» аналогична структуре в архитектуре «Тонкий клиент» (рисунок 2);



Рисунок 2 – Архитектура «Тонкий клиент»

3. Моделирование и реализация ПО

Архитектура ПО

Приложение реализовано по технологии клиент-сервер с использованием WCF. WCF — это среда выполнения и набор интерфейсов API для создания систем, отправляющих сообщения между службами и клиентами. Те же инфраструктура и интерфейсы API используются для создания приложений, обменивающихся данными с другими приложениями на данном компьютере или на компьютере, который находится в другой компании и доступ к которому можно получить через Интернет. Соответственно разрабатываются приложения: клиента и сервера. Приложение «Сервер» запускается в одном экземпляре. После запроса на подключение сервер устанавливает соединение с клиентом и обрабатывает его запросы. Приложение «Клиент» может быть запущено в нескольких экземплярах. Клиенты подключаются к серверу и посылают ему различные текстовые сообщения. Принятые от сервера сообщения отображаются в главном окне. Схема работы программного обеспечения представлена на рисунке 3.

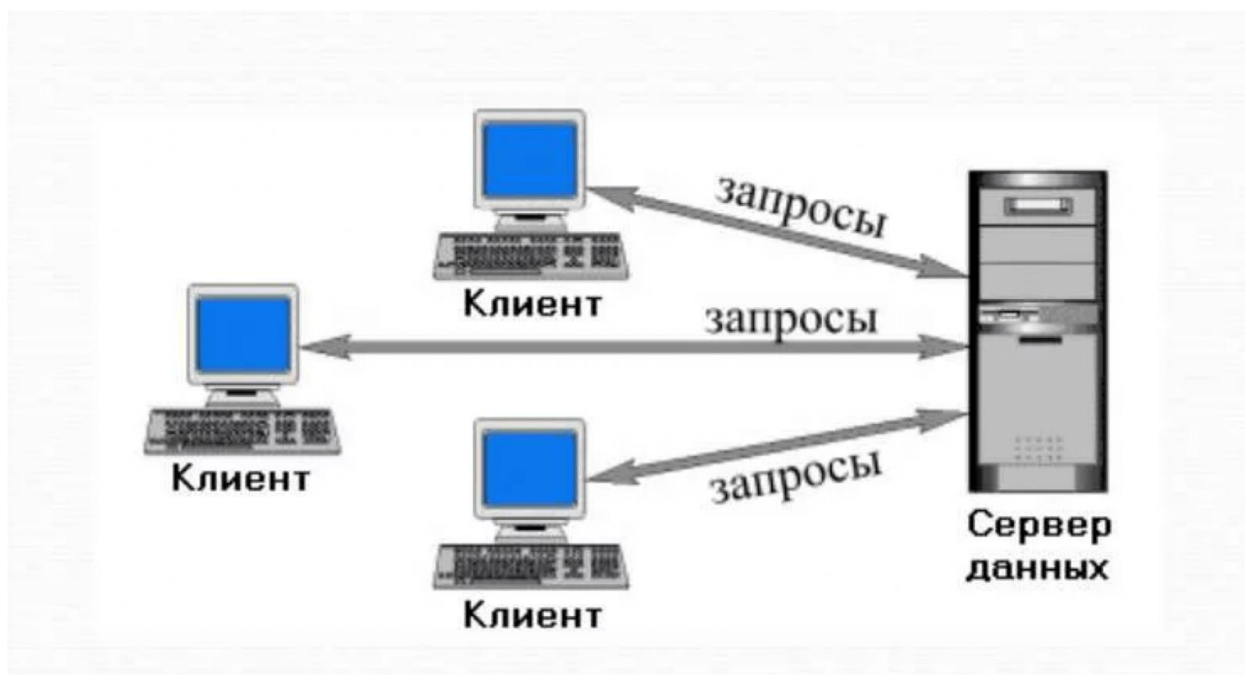


Рисунок 3 – схема ПО «Клиент - Сервер»

Описание функциональных модулей

Функциональный модуль «Сервер». Модуль отвечает за установление соединения с каждым новым клиентом. После соединения с клиентом модуль принимает сообщение от клиента, после чего он показывает это сообщение другим пользователям, подключенным к серверу. Также данный модуль демонстрирует, когда подключается новый пользователь или из «сети» выходит старый пользователь.

Сначала нам нужна библиотека классов .NET Framework

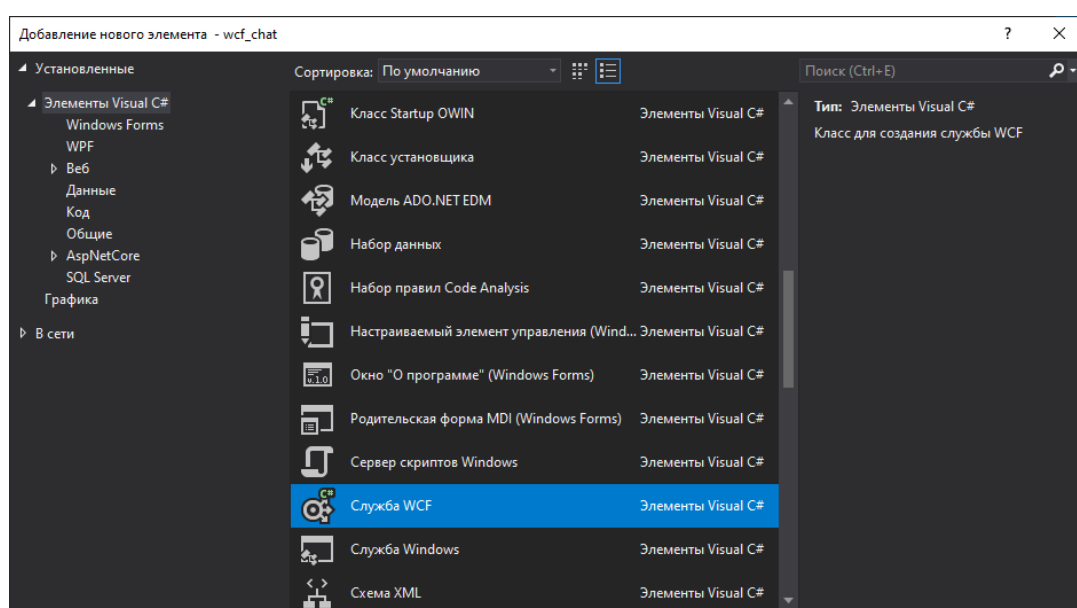


Рисунок 4

Полученный файл IService1 это описание того, что может делать наш сервис. Атрибут [ServiceContract] представляет собой договоренность о том, как клиент будет взаимодействовать с сервисом.

Как работает служба WCF с клиентом. Если мы вызываем какой-то метод, (например SendMsg), то клиент посылает нашему сервису это сообщение и далее ждет ответа от сервиса, что сообщение получено и даже если метод ничего не возвращает, клиент все равно будет заблокирован пока сервер ему не ответит. В некоторых случаях этот ответ ждать не требуется. Для этого атрибуту [OperationContract] нужно добавить параметр IsOneWay =

true. По умолчанию у всех атрибутов [OperationContract] есть данный параметр, но его значение = false.

```
namespace wcf_chat
{
    [ServiceContract(CallbackContract = typeof(IServerChatCallback))]
    public interface IService1
    {
        [OperationContract]
        int Connect(string name);

        [OperationContract]
        void Disconnect(int id);

        [OperationContract(IsOneWay = true)]
        void SendMsg(string msg, int id);
    }

    public interface IServerChatCallback
    {
        [OperationContract(IsOneWay = true)]
        void MsgCallback(string msg);
    }
}
```

Рисунок 5

А в классе Service1 выполняется реализация данного интерфейса

Создадим несколько методов. Все эти методы будут с оператором [OperationContract]

Первым будет метод Connect, он будет возвращать целочисленное значение int. С помощью данного метода мы будем подключаться к сервису.

В данном методе будут поля ID, Name, operationContext

```
public int Connect( string name)
{
    ServerUser user = new ServerUser
    {
        ID = nextId,
        Name = name,
        operationContext = OperationContext.Current
    };
    nextId++;
    SendMsg(" " + user.Name + " подключился к чату!", 0);
    users.Add(user);
    return user.ID;
}
```

```
}
```

Следующий метод Disconnect. Он будет вызываться, когда клиент хочет отключиться от сервиса. Данный метод не будет возвращать никаких параметров и будет принимать значение id клиента, которого нужно отключить:

```
public void Disconnect(int id)
{
    var user = users.FirstOrDefault(i => i.ID == id);
    if(user != null)
    {
        users.Remove(user);
        SendMsg(" " + user.Name + " покинул чат!",0);
    }
}
```

Следующий метод будет называться SendMsg. Он будет принимать строковое сообщение:

```
public void SendMsg(string msg, int id)
{
    foreach (var item in users)
    {
        string answer = DateTime.Now.ToShortTimeString();

        var user = users.FirstOrDefault(i => i.ID == id);
        if (user != null)
        {
            answer += " :" + user.Name + " ";
        }
        answer += msg;
    }
}
```

```
item.operationContext.GetCallbackChannel<IServerChatCallback>().MsgCallback(answer);}}
```

Метод callback

```
public void MsgCallback(string msg)
{
    lbChat.Items.Add(msg);
    lbChat.ScrollIntoView(lbChat.Items[lbChat.Items.Count-1]);
}
```

Чтобы сделать сервис общим для всех клиентов у WCF есть специальный функционал. Нужно указать поведение серверу - [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]. При указании параметра Single все клиенты при подключении будут подключаться к нашему хосту:

```
namespace wcf_chat
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
    public class Service1 : IService1
    {
        List<ServerUser> users = new List<ServerUser>();
        int nextId = 1;

        public int Connect( string name)
        {
            ServerUser user = new ServerUser
            {
                ID = nextId,
                Name = name,
                operationContext = OperationContext.Current
            };
            nextId++;
            SendMsg(" " + user.Name + " подключился к чату!", 0);
            users.Add(user);
            return user.ID;
        }

        public void Disconnect(int id)
        {
            var user = users.FirstOrDefault(i => i.ID == id);
            if(user != null)
            {
                // ...
            }
        }
    }
}
```

Рисунок 6

Для того, чтобы вызвать какое-то действие клиента со стороны сервера нам потребуется создать метод callback. На серверной части должен быть лишь его интерфейс, а реализация должна быть на стороне клиента. Реализуем интерфейс:

```
public interface IServerChatCallback
{
    [OperationContract(IsOneWay = true)]
    void MsgCallback(string msg);
}
```

В классе Service1 создадим список объектов ServerUser

```
List<ServerUser> users = new List<ServerUser>();
```

Добавим еще один класс ServerUser. Он будет являться «контейнером», в котором мы будем хранить три поля: id пользователя, его имя и поле operationcontecst. в данном поле будет содержаться информация о подключении нашего пользователя к сервису. Реализуем его:

```
3 namespace wcf_chat
4 {
5     class ServerUser
6     {
7         public int ID { get; set; }
8         public string Name { get; set; }
9         public OperationContext operationContext { get; set; }
10    }
11 }
12
```

Рисунок 7

Функциональный модуль «Клиент».

Модуль предоставляет интерфейс для взаимодействия с пользователем. Позволяет подключиться к серверу, ввести имя пользователя, необходимое для подключения. Выполним реализацию некоторых методов на стороне клиента:

Создадим переменную для генерации Id:

```
int ID;
```

Метод Connect():

```
void ConnectUser()
{
    if(!isConnected)
    {
        client = new Service1Client(new
System.ServiceModel.InstanceContext(this));
        ID = client.Connect(tbUserName.Text);
        tbUserName.IsEnabled = false;
        bConDiscon.Content = "Disconnect";
        isConnected = true;
    }
}
```

В данном методе мы создаем нового пользователя, у которого будут поля ID, Name, operationContecst.

После того, как пользователь будет подключен к серверу и его id будет установлен, данная переменная будет увеличена, а в общий чат будет выведено сообщение о подключении нового пользователя (рисунок 8):

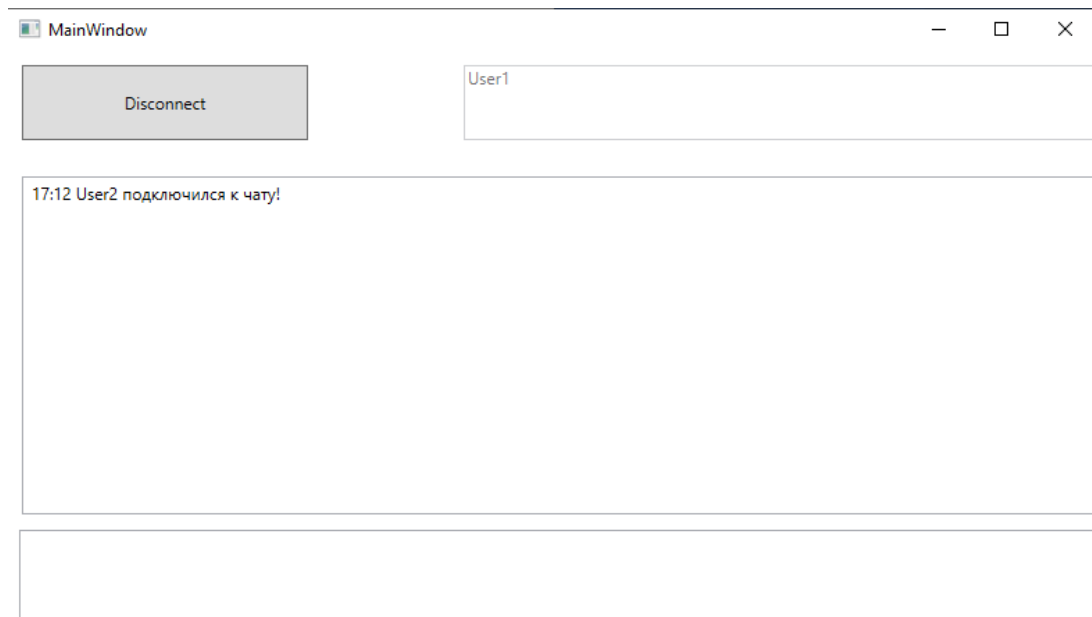


Рисунок 8 – сообщение о подключении пользователя

Метод Disconnect():

```
void DisconnectUser() {  
    if (isConnected)  
    {  
        client.Disconnect(ID);  
        client = null;  
        tbUserName.IsEnabled = true;  
        bConDiscon.Content = "Connect";  
        isConnected = false;  
    }  
}
```

В данном методе мы будем находить id пользователя, который хочет отключиться и отключать его от сервера (рисунок 9):

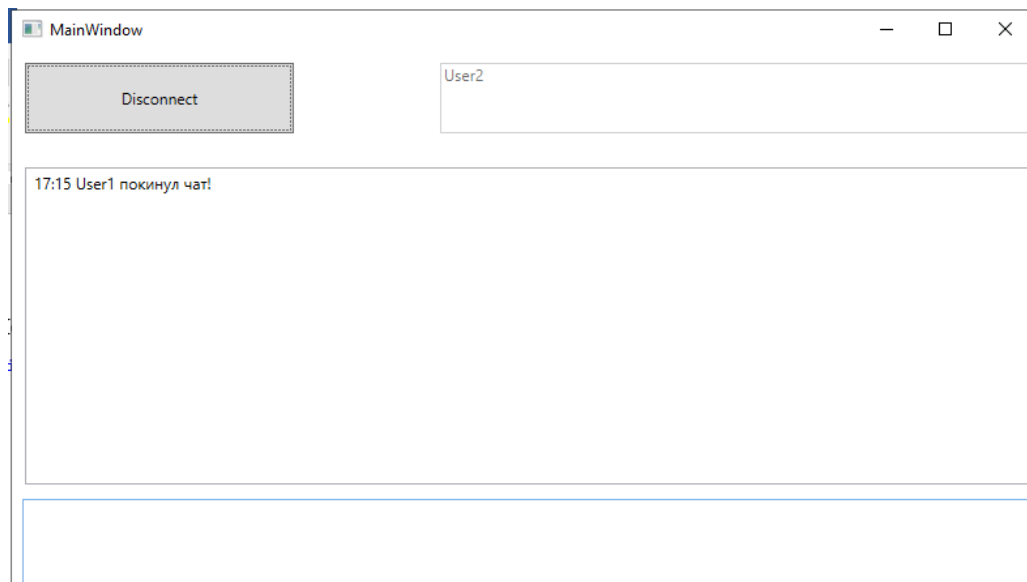


Рисунок 9 – сообщение о выходе пользователя из сети

Метод SendMsg():

```
private void tbMessage_KeyDown(object sender, KeyEventArgs e)
{
    if(e.Key == Key.Enter)
    {
        if(client != null)
        {
            client.SendMsg(tbMessage.Text, ID);
            tbMessage.Text = string.Empty;
        }
    }
}
```

В данном методе мы будем перебирать всех пользователей и будем формировать сообщение, которое будет ответом от сервера всем пользователям. В этом ответе будет содержаться: дата (время отправки текущего сообщения); имя пользователя, отправившего сообщение; сам текст сообщения (рисунок 10)

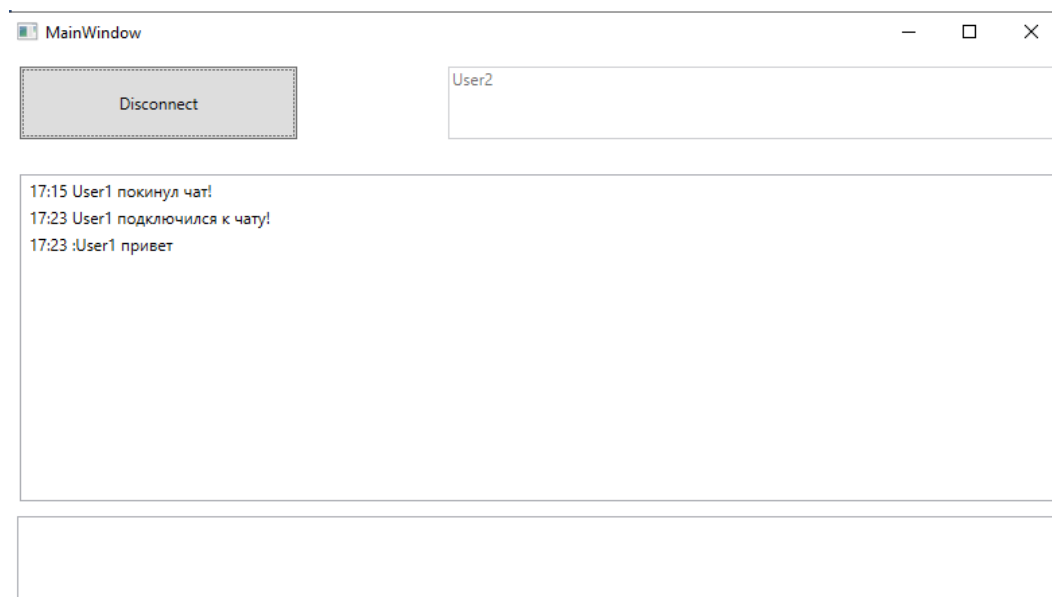


Рисунок 10 – отправка текстового сообщения

Интерфейс пользователя

Для взаимодействия с пользователем в приложении используется оконный интерфейс (рисунок 11). Благодаря данному окну пользователь может выбрать имя, подключиться к общей сети пользователей, получать и отправлять сообщения или выйти (рисунок 5).

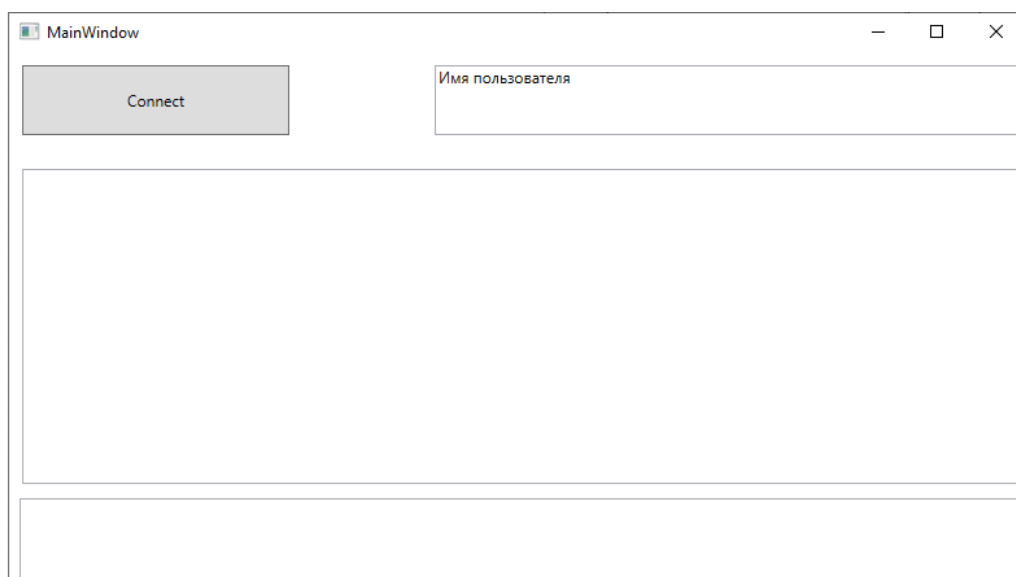


Рисунок 11 – интерфейс пользователя

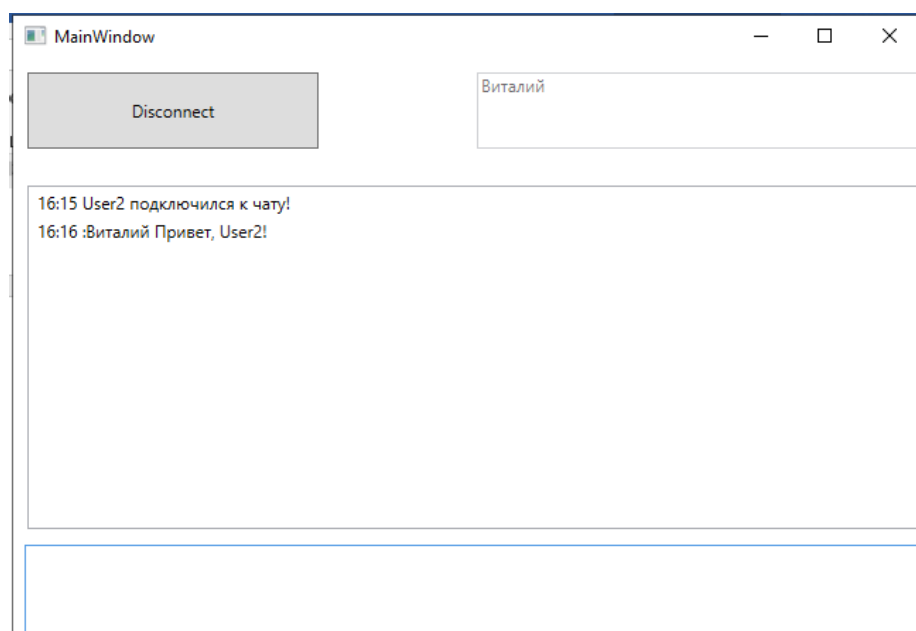


Рисунок 12 – отправка сообщения

Организация сетевого взаимодействия

Сетевое взаимодействие происходит посредством передачи и приема данных через службы WCF, связывающие клиента и сервера. После установления соединения с клиентом, сервер начинает отображать клиенту сообщения, которые пишут другие пользователи. (Рисунок 13)

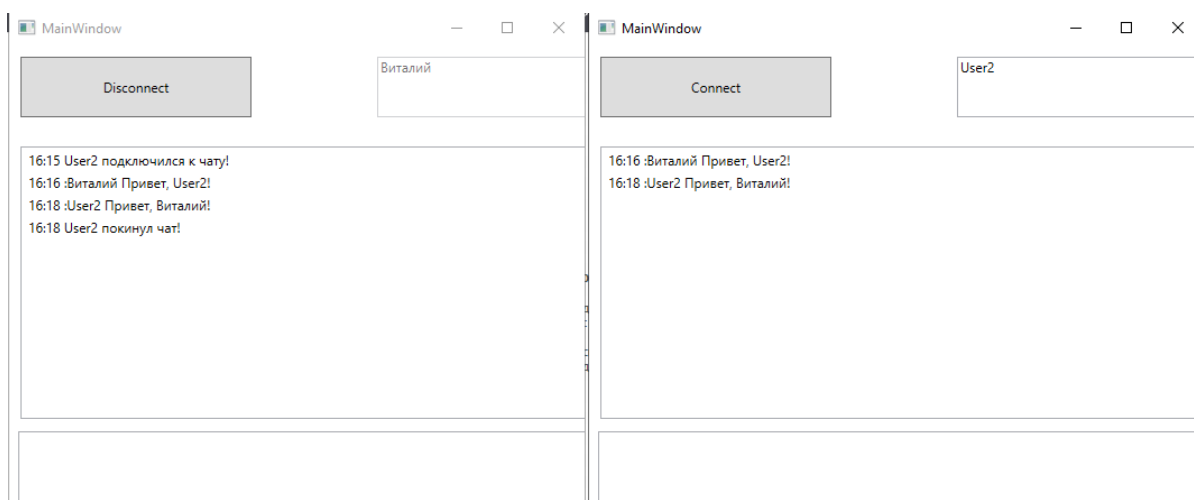


Рисунок 13 – получение сообщения

Хост. Для создания хоста подойдет обычное консольное приложение. Добавим его в это же решение. Затем в ссылках нужно добавить ссылку System.ServiceModel. также нужно добавить ссылку на dll в которую будет собран проект:

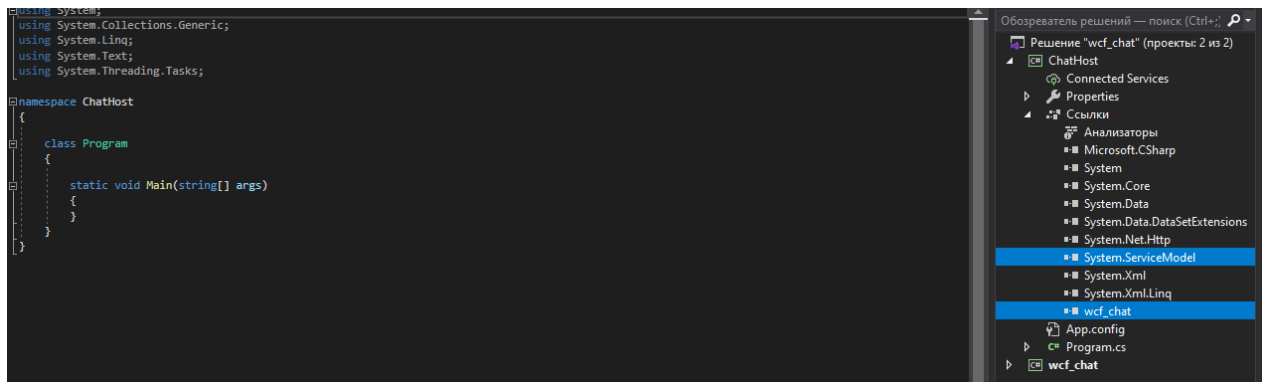


Рисунок 14

Что требуется от хоста? У хоста есть файл config, в нем мы будем описывать конфигурацию хоста. Данный файл нужен для того, чтобы осуществить настройку нашего хоста: где он будет работать, на каких портах, какие протоколы будет использовать, какой IP будет использовать, а также для того, чтобы организовать пересылку метаданных от хоста к клиенту.

Сервис будет работать по протоколу TCP, а обмен метаданными будет работать по HTTP

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.7.2" />
  </startup>

  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="mexBeh">
```

```

        <serviceMetadata httpGetEnabled="true"
httpsGetEnabled="true" />
        <serviceDebug
includeExceptionDetailInFaults="false" />
    </behavior>
</serviceBehaviors>
</behaviors>
<services>
    <service name="wcf_chat.Service1">
        <endpoint address="" binding="netTcpBinding"
contract="wcf_chat.IService1">

            </endpoint>
            <endpoint address="mex"
binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
            <baseAddresses>
                <add
baseAddress="http://localhost:8301" />
                <add
baseAddress="net.tcp://localhost:8302" />
            </baseAddresses>
        </host>
    </service>
</services>
</system.serviceModel>
</configuration>

```

Перейдем к реализации хоста

```

namespace ChatHost
{
    class Program
    {
        static void Main(string[] args)
        {

```

```

        using (var host = new
ServiceHost(typeof(wcf_chat.Service1)))
        {
            host.Open();
            Console.WriteLine("Хост запустился!");
            Console.ReadLine();
        }
    }
}

```

Проверка работы хоста (рисунок 15):

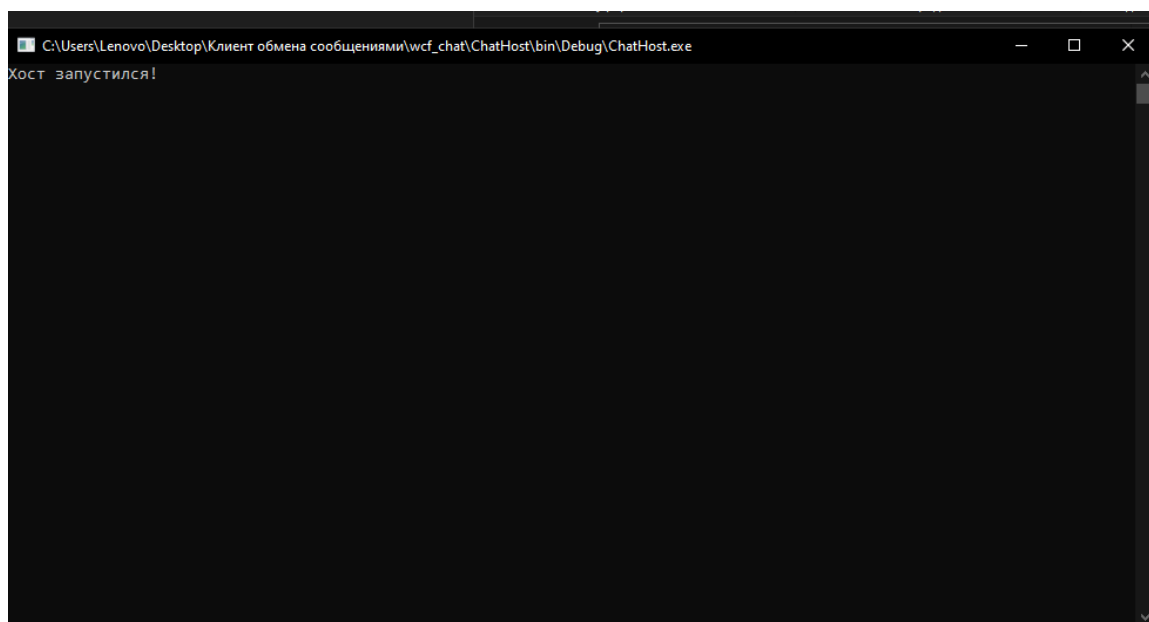


Рисунок 15 – проверка работы хоста

Руководство пользователя

Для удобства пользователю предоставляется русскоязычное меню. Пользователю нужно заполнить текстовое поле «Имя пользователя», после чего нажать кнопку «Connect». После этого пользователь будет подключен к серверу и сможет отправлять сообщения. Для этого нужно нажать на текстовое поле внизу формы, набрать сообщение и нажать клавишу «Enter». Сообщение будет отправлено. В случае, если пользователю захочется выйти он может нажать на кнопку «Disconnect» и закрыть окно, либо просто закрыть окно, программы отключит пользователя от сервера автоматически.

4.3 Заключение

В ходе выполнения курсовой работы были созданы приложения клиента и сервера, демонстрирующие использование технологии «клиент-сервер» на примере удаленного генератора псевдослучайных последовательностей.

Программа написана на языке C# в среде программирования Visual Studio 2019. Тестирование показало правильность работы приложения. Разработка и тестирование производились в ОС Windows 10.

Программа представляет собой удобный пользовательский интерфейс, интуитивно понятный для любого пользователя, программе не требуется установка.

5. Список используемых источников

1. <https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/whats-wcf>
2. Мюллер Д. П. С# для чайников
3. Прайс. С# 7 и .NET Core. Кросс-платформенная разработка для профессионалов
4. Троелсен и Джепикс. Язык программирования С# 7 и платформы .NET и .NET Core

6. Приложение

Приложение А

Приложение А.1

Файл Service1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace wcf_chat
{
    [ServiceBehavior(InstanceContextMode =
InstanceContextMode.Single)]
    public class Service1 : IService1
    {

        List<ServerUser> users = new List<ServerUser>();
        int nextId = 1;

        public int Connect( string name)
        {
            ServerUser user = new ServerUser
            {
                ID = nextId,
                Name = name,
                operationContext = OperationContext.Current
            };
            nextId++;
            SendMsg(" " + user.Name + " подключился к чату!", 0);
            users.Add(user);
            return user.ID;
        }

        public void Disconnect(int id)
        {
            var user = users.FirstOrDefault(i => i.ID == id);
            if(user != null)
            {
                users.Remove(user);
                SendMsg(" " + user.Name + " покинул чат!", 0);
            }
        }

        public void SendMsg(string msg, int id)
        {

```

```

        foreach (var item in users)
        {
            string answer =
DateTime.Now.ToShortTimeString();

            var user = users.FirstOrDefault(i => i.ID ==
id);

            if (user != null)
            {
                answer += " :" + user.Name + " ";
            }
            answer += msg;

            item.operationContext.GetCallbackChannel<IServerChatCallback>().
MsgCallback(answer);
        }
    }
}

```

Приложение A.2

Файл ServerUser.cs

```

using System.ServiceModel;

namespace wcf_chat
{
    class ServerUser
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public OperationContext operationContext { get; set; }
    }
}

```

Приложение A.3

Файл IService1.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace wcf_chat
{
    [ServiceContract(CallbackContract =
typeof(IServerChatCallback))]
    public interface IService1
    {

```

```

        [OperationContract]
        int Connect(string name);

        [OperationContract]
        void Disconnect(int id);

        [OperationContract(IsOneWay = true)]
        void SendMsg(string msg, int id);

    }

    public interface IServerChatCallback
    {
        [OperationContract(IsOneWay = true)]
        void MsgCallback(string msg);
    }
}

```

Приложение A.4

Файл Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;

namespace ChatHost
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var host = new
ServiceHost(typeof(wcf_chat.Service1)))
            {
                host.Open();
                Console.WriteLine("Хост запустился!");
                Console.ReadLine();
            }
        }
    }
}

```

Приложение A.5

Файл MainWindow.cs

```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using ChatClient1.ServiceChat;
namespace ChatClient1
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window,
ChatClient1.ServiceChat.IService1Callback
    {
        bool isConnected = false;
        Service1Client client;
        int ID;

        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender,
RoutedEventArgs e)
        {
        }

        void ConnectUser()
        {
            if(!isConnected)
            {
                client = new Service1Client(new
System.ServiceModel.InstanceContext(this));
                ID = client.Connect(tbUserName.Text);
                tbUserName.IsEnabled = false;
                bConDiscon.Content = "Disconnect";
                isConnected = true;
            }
        }

        void DisconnectUser() {
            if (isConnected)
            {

```

```

        client.Disconnect(ID);
        client = null;
        tbUserName.IsEnabled = true;
        bConDiscon.Content = "Connect";
        isConnected = false;
    }
}

private void bConDiscon_Click(object sender,
RoutedEventArgs e)
{
    if(!isConnected)
    {
        ConnectUser();
    }
    else
    {
        DisconnectUser();
    }
}

public void MsgCallback(string msg)
{
    lbChat.Items.Add(msg);

lbChat.ScrollIntoView(lbChat.Items[lbChat.Items.Count-1]);
}

private void Window_Closing(object sender,
System.ComponentModel.CancelEventArgs e) // в случае закрытия
окна
{
    DisconnectUser();
}

private void tbMessage_KeyDown(object sender,
KeyEventArgs e)
{
    if(e.Key == Key.Enter)
    {
        if(client != null)
        {
            client.SendMsg(tbMessage.Text, ID);
            tbMessage.Text = string.Empty;
        }
    }
}
}
}

```

Приложение A.6

Файл app.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.7.2" />
  </startup>

  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="mexBeh">
          <serviceMetadata httpGetEnabled="true"
httpsGetEnabled="true" />
          <serviceDebug
includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="wcf_chat.Service1"
behaviorConfiguration="mexBeh">
        <endpoint address="" binding="netTcpBinding"
contract="wcf_chat.IService1">

          </endpoint>
          <endpoint address="mex"
binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add
baseAddress="http://localhost:8301" />
            <add
baseAddress="net.tcp://localhost:8302" />
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>

```

Приложение В

Результат работы программы Приложение В.1



Рисунок 16 – запуск хоста

Приложение В.2

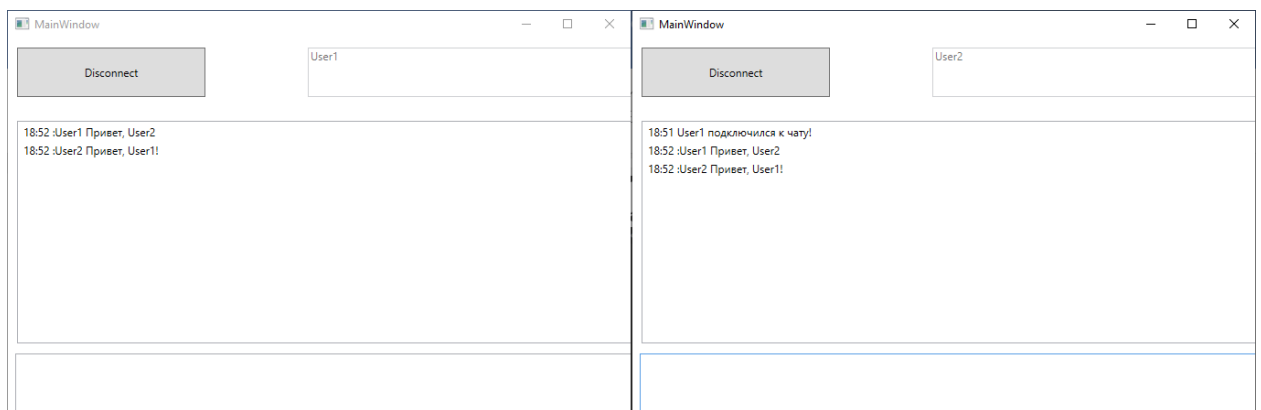


Рисунок 17– обмен сообщениями между двумя клиентами

Приложение В.3

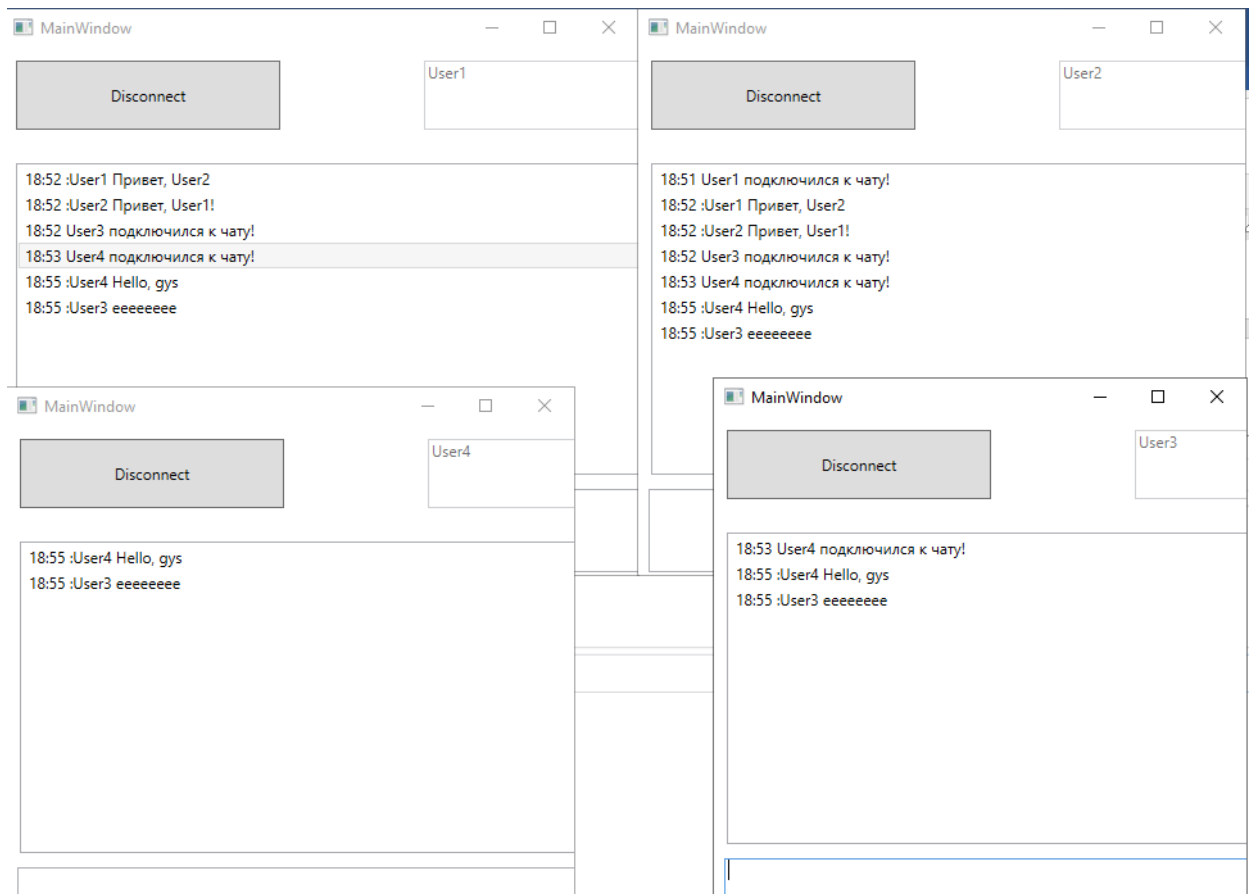


Рисунок 18 – обмен сообщениями между четырьмя клиентами