ПРАКТИЧЕСКАЯ РАБОТА № 19

(4 академических часа)

Тема: Работа с сокетами TCP в .NET

Цель работы:

1. Понять основы разработки сетевых приложений в Visual Studio.
2. Научиться разрабатывать клиент-серверной приложение, используя сокеты.

Задание: создать клиент-серверное приложение, осуществляющее функции обмена сообщениями в виде чата.

Методика выполнения задания:

В практической работе необходимо разработать два приложения. Первое – сервер, реализующий комнату чата для обмена текстовыми сообщениями. Второе – клиент для работы с чатом.

Серверное приложение должно выполнять следующие функции:

- принимать входящие соединения;
- вести список пользователей;
- передавать пользователям информацию о новых участниках, их сообщениях и информацию об уходящих участниках.

Клиентское приложение должно:

- подключаться к серверу;
- передавать серверу имя пользователя, его сообщения и сигнализировать о выходе из чата;
- получать от сервера информацию о действиях других участников чата.

Сервер и клиент должны обмениваться короткими текстовыми сообщениями разных типов.

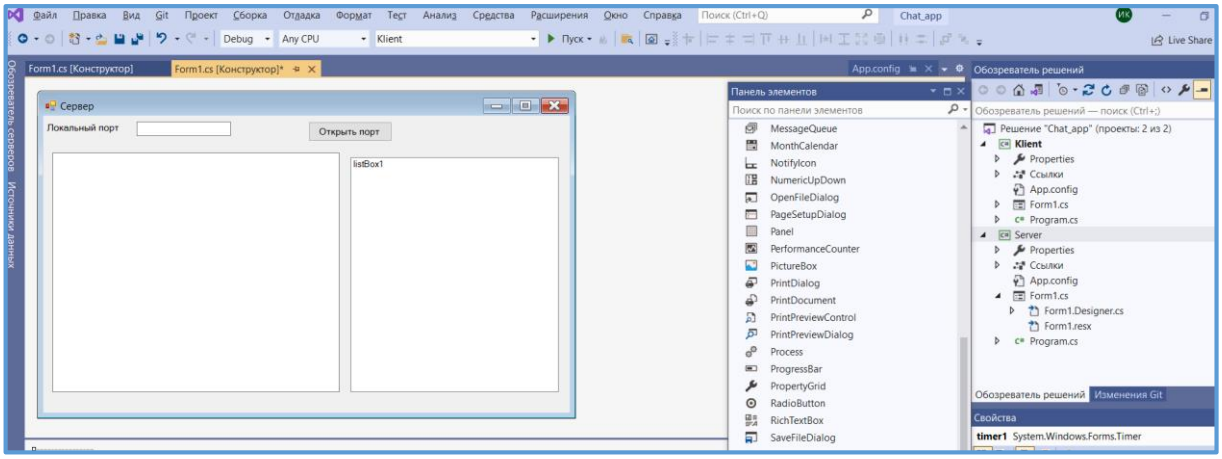
Клиент может посылать серверу следующие команды:

- Клиент <имя> - задать для пользователя имя, где <имя> - произвольные текст;
- Сообщение <сообщение> - написать сообщение в чат;
- Отключение – пользователь покидает чат.

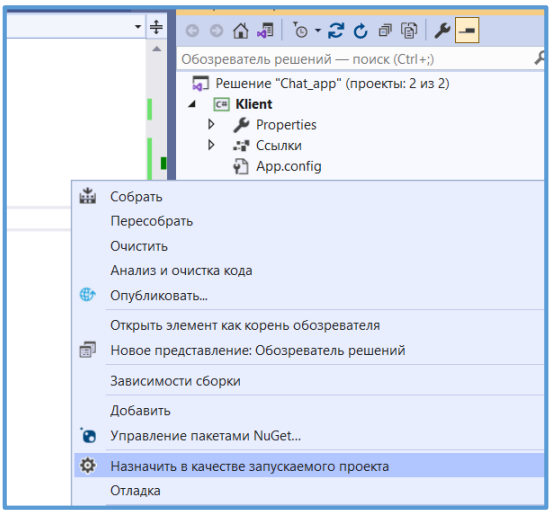
После получения команды от клиента сервер выполняет нужное действие и всем остальным участникам чата (кроме того, кто инициировал событие) высылает информационные сообщения:

- Клиент <имя> - в чате новый участник <имя>;
- Сообщение <имя: сообщение> - новое сообщения от участника;
- Отключение <имя> – пользователь покидает чат.

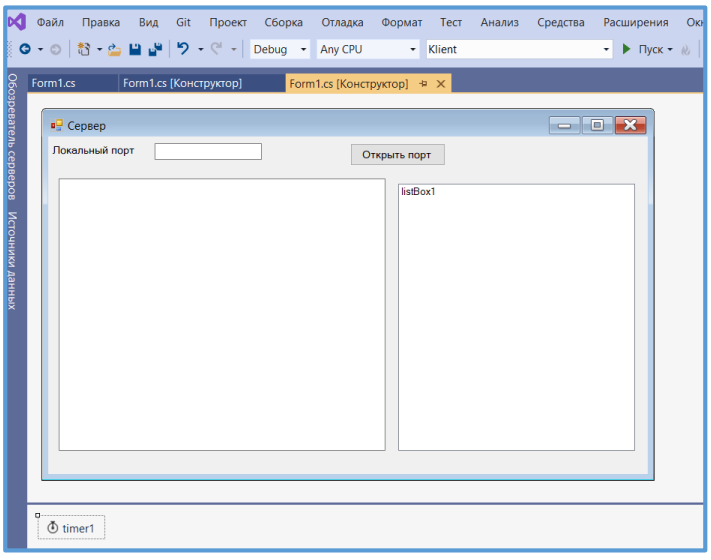
1. Открываем Microsoft Visual Studio и создаем в одном решении 2 проекта Windows Forms: Sever, который будет выполнять функции сервера, и Klient, который будет выполнять функции клиента.



2. Один из проектов является активным, а другой пассивным. Чтобы сменить активный проект надо в обозревателе решений щелкнуть правой кнопкой на строке с проектом и в меню выбрать пункт «Назначить запускаемым проектом».



3. Для создания интерфейса Сервера добавляем на форму элементы:
   - текстовые поля textBox1 (локальный порт) и textBox2 (поле текстового вывода);
   - кнопка button1 (открыть порт);
   - список listBox (класс ListBox) для отображения списка пользователей;
- таймер timer1 (класс Timer).



4. *Создание исходного кода сервера.*

4.1. Подготовка переменных для хранения данных

Сначала необходимо подключить пространства имен для работы с сокетами:

```
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;
using System.Net;
using System.Net.Sockets;

namespace Chat_app
{
```

Серверному приложению понадобиться хранить информацию о текущих клиентах. Во-первых, для каждого клиента надо хранить информацию о сокете, через который с ним происходит общение. Кроме того, нужно хранить имя, которым решил назваться пользователь.

Для этого в программе создадим класс *ClientInfo*. В этом классе укажем два поля: **socket** типа Socket и *name* типа string. Этот класс содержит один метод ToString, который возвращает текст в виде: <имя> (<адрес>: <порт>). Этот метод нужен для адекватного отображения перечня клиентов в поле-списке listBoxClients.

Для отслеживания входящих соединений в классе формы объявим переменную listener типа TcpListener. Этот класс является удобной «надстройкой» над классом Socket, упрощающей написание кода.

Для хранения списка пользователей также объявим переменную *clients* типа List<ClientInfo> (означает, что в переменной clients будет храниться список объектов типа ClientInfo.

```
namespace Chat_app
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        Ссылок: 8
        class ClientInfo   // для хранения информации о клиентах
        {
            public Socket socket;  // сокет
            public string Name;    // имя клиента
            Ссылок: 0
            public override string ToString() // метод для преобразования объекта в текстовую строку
            {
                return Name + "("+ socket.RemoteEndPoint+")";
            }

        }
        TcpListener listener;  // объект для приема входящих TCP-соединений
        List<ClientInfo> clients;  // переменная clients - для хранения списка объектов ClientInfo
        ссылка: 1
        public Form1()
        {
```

4.2. Создание сокета входящих соединений

Для кнопки *button1* создадим обработчик события Click. Этот обработчик будет выполнять следующие действия:

- создавать объект TcpListener и связывать его с портом, указанным в поле textBoxLocalPort;
- начинать прослушивание входящих соединений;
- создавать пустой список clients для хранения данных о клиентах;
- включать таймер для обработки соединений

```
private void button1_Click(object sender, EventArgs e) // открыть порт
{
    try
    {
        int localPort = int.Parse(textBox1.Text);  // числовое значение порта
        // создание пары порт+хост для открытия сокета
        IPEndPoint localPoint = new IPEndPoint(IPAddress.Any,localPort);
        // создание объекта TCPListener
        listener = new TcpListener(localPoint);
        listener.Start();   // запуск сервера на прослушивание
        clients = new List<ClientInfo>();  // создание пустого списка клиентов
        timer1.Enabled = true; // запуск таймера
        textBox2.AppendText("Порт открыт "+textBox1.Text + "\r"+"\n");
    }
    catch (Exception exp)
    {
        textBox2.AppendText(exp.Message+"\r"+"\n");
    }
}
```

4.3. Обработка соединений

```
private void timer1_Tick(object sender, EventArgs e)  // обработчик таймера
{
    try
    {
        CheckListener();  // проверка новых подключений
        for (int i = clients.Count - 1; i >= 0; i--)
        {
            ClientInfo client = clients[i];
            if (client.socket.Available > 0)
            {
                byte[] data = new byte[client.socket.Available];
                int data_size = client.socket.Receive(data);
                string text_data = Encoding.UTF8.GetString(data,0,data_size);
                DoClient(client, text_data);
            }
        }
    }
    catch (Exception exp)
    {
        textBox2.AppendText(exp.Message + "\r" + "\n");
    }
}
```

4.4. Прием входящих соединений (метод CheckListener())

```
private void CheckListener()  // метод проверки новых соединений
{
    if (listener.Pending())   // есть новые соединения
    {
        ClientInfo newClient = new ClientInfo();
        newClient.socket = listener.AcceptSocket();
        clients.Add(newClient);
        textBox2.AppendText("Пользователь "+newClient.socket.RemoteEndPoint+ " подключился" +"\r"+ "\n");
    }

}
```

Этот метод выполняет следующие действия:

- проверяет наличие новых соединений, вызывая метод listener.Pending;
- в случае их наличия создает новый объект ClientInfo для хранения информации о новом соединении;
- создает сокет методом listener.AcceptSocket (принять соединение);
- добавляет в список client информацию о новом клиенте.

4.5. Обработка сообщений от клиента (метод DoClient())

```
private void DoClient (ClientInfo client, string text_data)
    // метод обработки данных от клиента
{
    if (text_data.StartsWith("Клиент "))
    {
        client.Name = text_data.Substring(6); // сохранение имени клиентов поле Name объекта
        listBox1.Items.Add(client);  // добавление клиента в список клиентов
        SendToClients("Новый клиент" +client.Name, client);
        textBox2.AppendText("Пользователь "+ client.socket.RemoteEndPoint +" с именем "+ client.Name +"\r"+"\n");
    }
    if (text_data == "Отключение ")
    {
        SendToClients("Отключение "+client.Name, client);
        textBox2.AppendText("Пользователь " + client.socket.RemoteEndPoint + " вышел из чата" + "\r" + "\n");
       client.socket.Shutdown(SocketShutdown.Both); // отключение сокета на прем и отправку сообщений
        client.socket.Close(); // закрытие соединения
        listBox1.Items.Remove(client);  // удаление из списка на форме
        clients.Remove(client);  // удаление из объекта список
    }
    if (text_data.StartsWith("Сообщение "))
    {
        string message = text_data.Substring(9);
        SendToClients("Сообщение от клиента "+ client.Name +": "+message,client);
        textBox2.AppendText(client.Name+": "+message+"\r"+"\n");
    }
}
```

Метод проверяет, является ли сообщение одной из команд: Клиент, Сообщение или Отключение. Для проверки используется метод *text_data.StartsWith*, который проверяет, начинается ли строка text_data с указанной последовательности символов.

Для команды Клиент сервер выполняет следующие действия:

- сохраняет в поле client.Name полученное имя, используя метод text_data.Substring, которая выдает все символы в строке text_data начиная с указанного номера и до конца;
- добавляет в поле listBox1 новый элемент;
- с помощью метода SendToClients посылает всем клиентам сообщение Новый клиент.

Для команды Отключение сервер выполняет следующие действия:

- с помощью метода SendToClients посылает всем клиентам сообщение exit;
- разрывает соединение;
- удаляет элемент client из поля listBoxё и списка clients.

Для команды Сообщение сервер с помощью метода SendToClients посылает всем клиентам сообщение message.
Для рассылки сообщений клиентами запускается метод SendToClients, приведенный ниже. Этот метод высылает команду command всем клиентам, но не клиенту, указанному в параметре exceptOf. Метод можно разместить сразу под методом DoClient.

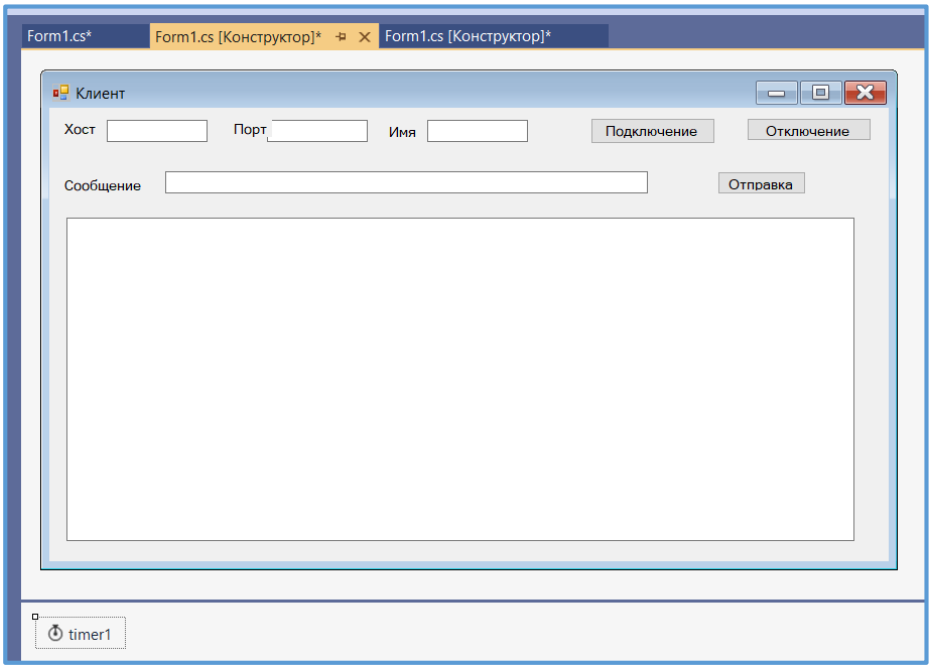
```
private void SendToClients(string command, ClientInfo exceptOf)
    // отправка сообщения всем клиентам, кроме клиента, указанного в exceptOf
{
    for(int i=0; i<clients.Count;i++)
    {
        ClientInfo client = clients[i];
        if (client !=exceptOf)
        {
            try
            {
                byte[] data = Encoding.UTF8.GetBytes(command);
                client.socket.Send(data);
            }
            catch (Exception exp)
            {
                textBox2.AppendText(exp.Message+"\n");
            }
        }

    }
}
```

5. *Создание интерфейса клиентского приложения*

Интерфейс клиентского приложения включает следующие элементы:

- текстовые поля textBox1 (хост), textBox2 (порт), textBox2 (имя), textBox4 (сообщение) и textBox5 (поле текстового вывода);
- кнопки button1 (подключение), button2 (отключение) и button3 (отправка);
- для кнопки «Отключение» задайте свойство Enabled (доступно) равным false.
- таймер timer1 (класс Timer).



6. *Создание исходного кода клиента*

Вначале необходимо, как и ранее, подключить к файлу исходного кода формы две библиотеки: System.Net и System.Net.Sockets.

В классе формы объявим переменную socket для хранения информации о сокете. Также создадим вспомогательный метод SendToServer, который будет отправлять строку command.

```
namespace Klient
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        Socket socket;
        Ссылок: 3
        private void SendToserver(string command)
        {
            byte[] data = Encoding.UTF8.GetBytes(command);
            socket.Send(data);
        }
        ссылка: 1
        public Form1()
        {
            InitializeComponent();
        }
```

Для кнопки button1 создадим обработчик события Click. Этот обработчик будет выполнять следующие действия:

- создавать объект Socket, настроенный на протокол TCP;
- подключаться к серверу по данным из textBox1 и textBox2;
- отправлять серверу команду name с данными из textBox3;
- включить таймер для обработки входящих данных и управления кнопками Соединение/Отключение.

```
private void button1_Click(object sender, EventArgs e)  // подключение
{
    try
    {
        socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        socket.Connect(textBox1.Text,Int32.Parse(textBox2.Text));
        SendToserver("Клиент "+textBox3.Text);
        timer1.Enabled = true; // включение таймера
        textBox5.AppendText("Подключение к "+textBox1.Text+": "+textBox2.Text+"\r"+"\n");
        button1.Enabled = false;
        button2.Enabled = true;
    }
    catch (Exception exp)
    {
        textBox5.AppendText(exp.Message+"\n");
    }
}
```

Для кнопки button2создадим обработчик события Click. Этот обработчик выполняет следующие действия:

- посылает серверу команду Отключение;
- закрывает соединение;
- останавливает таймер и меняет состояние кнопок.

```
private void button2_Click(object sender, EventArgs e) // закрытие соединения
{
    try
    {
        SendToserver("Отключение ");
        socket.Shutdown(SocketShutdown.Both); // отключение сокета для отправки и приема сообщений
        socket.Close();
        timer1.Enabled = false;   // отключение таймера
        button1.Enabled = true;
        button2.Enabled = false;
        textBox5.AppendText("Отключено" +"\r"+"\n");

    }
    catch (Exception exp)
    {
        textBox5.AppendText(exp.Message + "\n");
    }
}
```

Код обработчика кнопки button3 отправляет серверу команду Сообщение.

```
private void button3_Click(object sender, EventArgs e)  // отправка сообщений
{
    try
    {
       SendToserver("Сообщение "+textBox4.Text);
        textBox5.AppendText(textBox3.Text + ": " + textBox4.Text + "\r"+"\n");

    }
    catch (Exception exp)
    {
        textBox5.AppendText(exp.Message + "\n");
    }
}
```

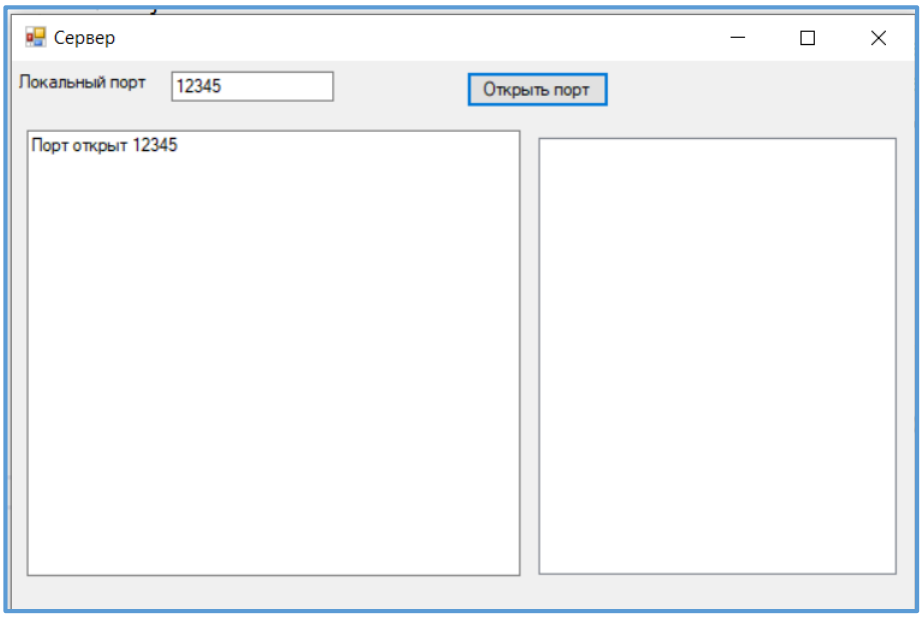
Последний и наиболее сложный обработчик – это обработчик таймера timer1. Этот обработчик очень похож на метод DoClient у сервера: также сначала определяет тип полученного сообщения и в зависимости от этого выводит на экран разную информацию:

```
private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        if(socket.Available>0)  // если есть новые данные
        {
            byte[] data = new byte[socket.Available]; // создание буфера для приема сообщений
            int data_size = socket.Receive(data); // получение данных
            string text_data = Encoding.UTF8.GetString(data,0,data_size);
            if(text_data.StartsWith("Новый клиент "))
            {
                textBox5.AppendText(text_data.Substring(13) + " в чате" +"\r"+"\n");
            }
            if (text_data.StartsWith("Отключение "))
            {
                textBox5.AppendText(text_data.Substring(11) + " вышел из чата"+"\r"+"\n");
            }
            if (text_data.StartsWith("Сообщение "))
            {
                textBox5.AppendText(text_data.Substring(21) + "\r"+"\n");
            }
        }
    }
    catch (Exception exp)
    {
        textBox5.AppendText(exp.Message + "\n");
    }
```

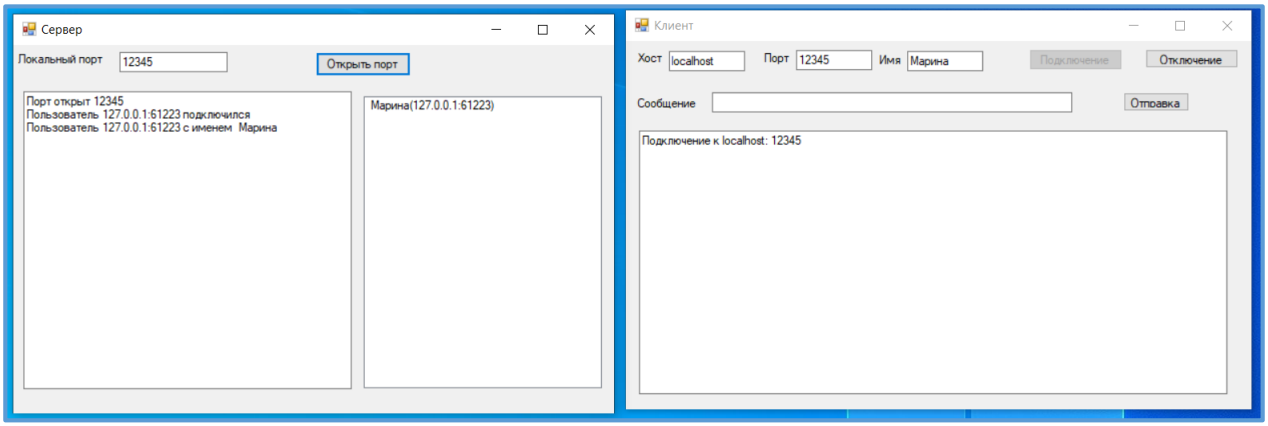
Проблемы не найдены

7. *Тестирование программы*

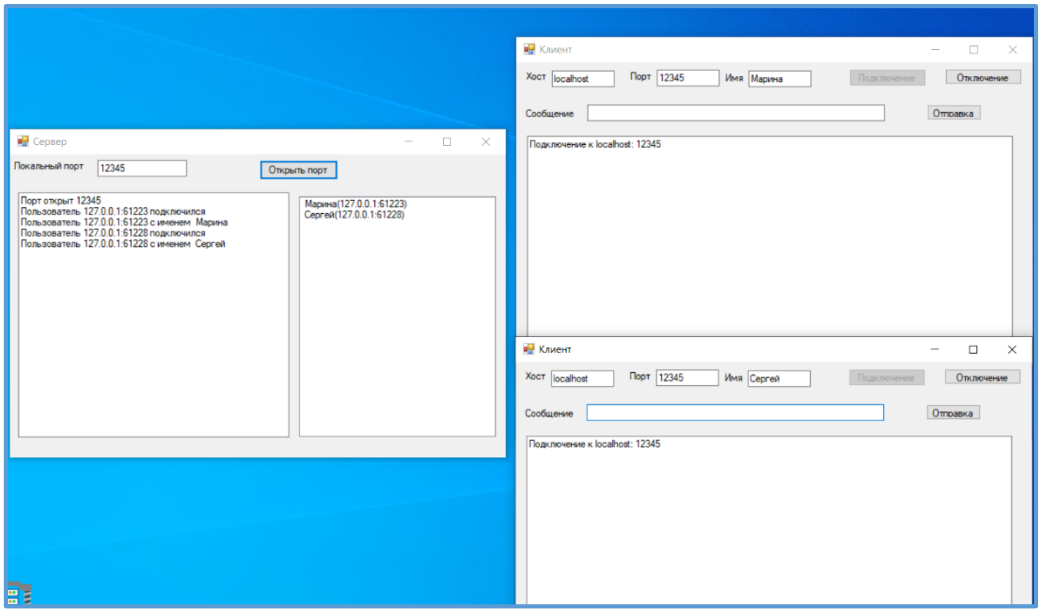
1) Сделайте активным проект для сервера и запустите его в режиме без отладки. Откройте порт 12345.



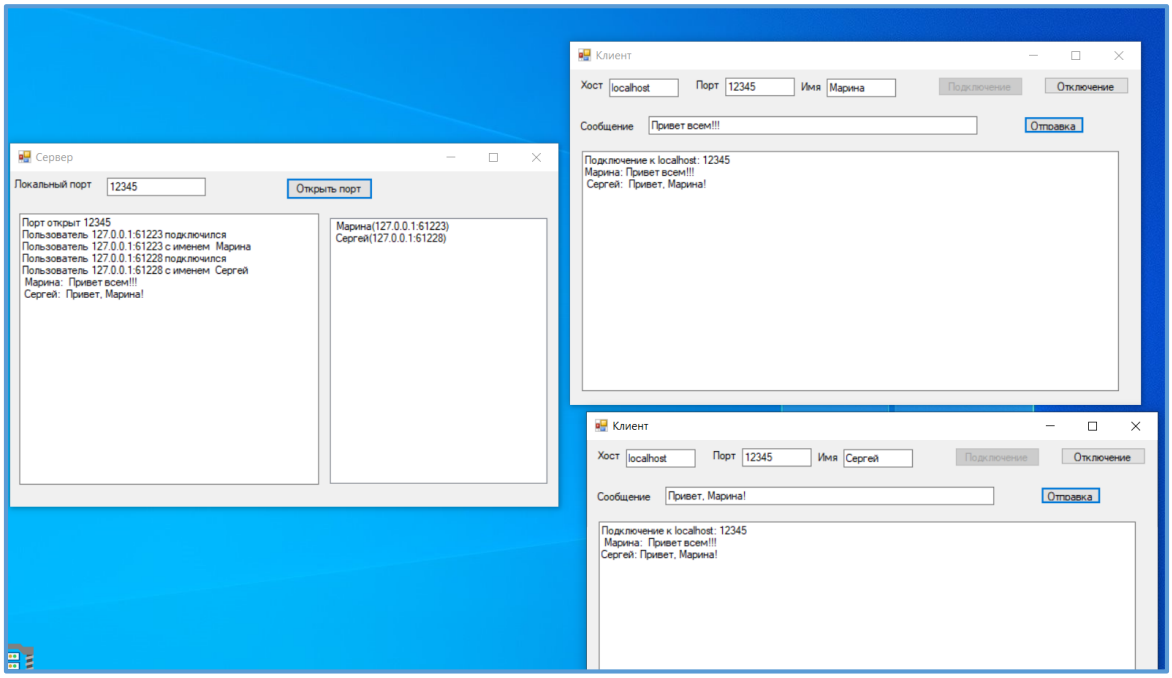
2) Сделайте активным проект клиента и запустите два его экземпляра без отладки.



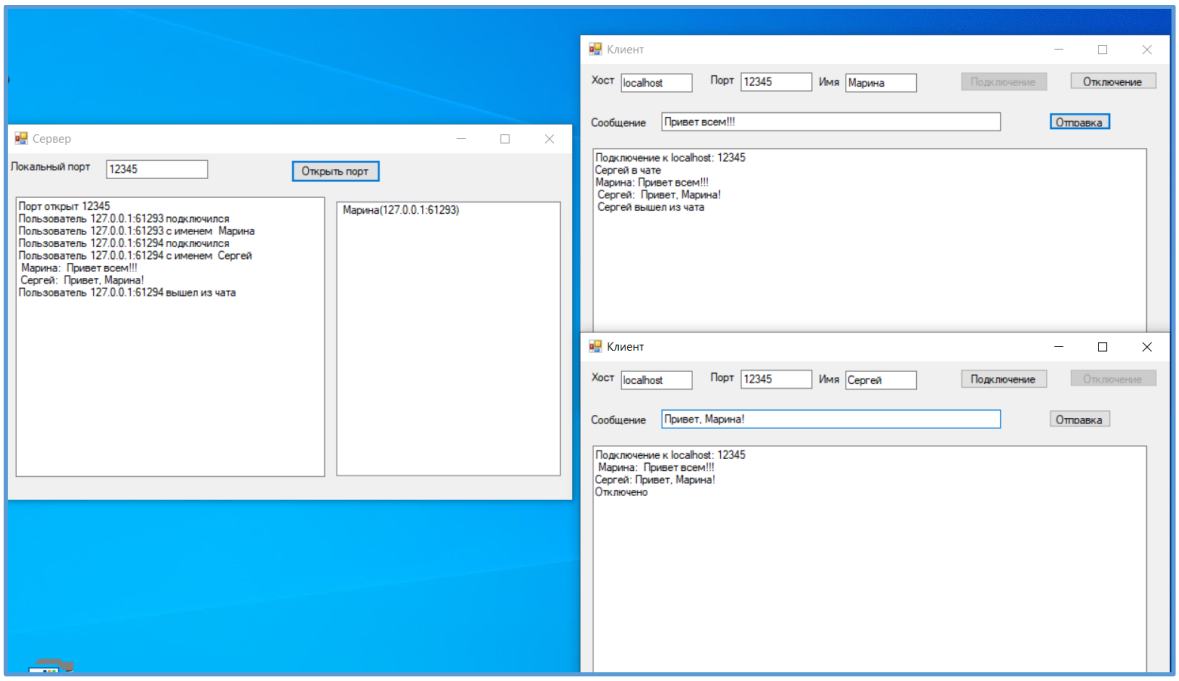
3) Последовательно подключите оба клиента к серверу.



4) Попробуйте отправку сообщений.



5) Для обоих клиентов разорвите соединение.



6) Попробуйте зайти на сервера на других компьютерах. Пусть кто-то еще зайдет на ваш сервер.