

Лабораторная работа №2 – часть 3. Работа с сокетами TCP в .NET

Целью этой лабораторной является создание двух 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.

Ниже приводится синтаксис и семантика протокола их взаимодействия.

Клиент может посылать серверу следующие команды:

- name <имя> - задать для пользователя имя, где <имя> - произвольные текст;
- message <сообщение> - написать сообщение в чат;
- quit – пользователь покидает чат.

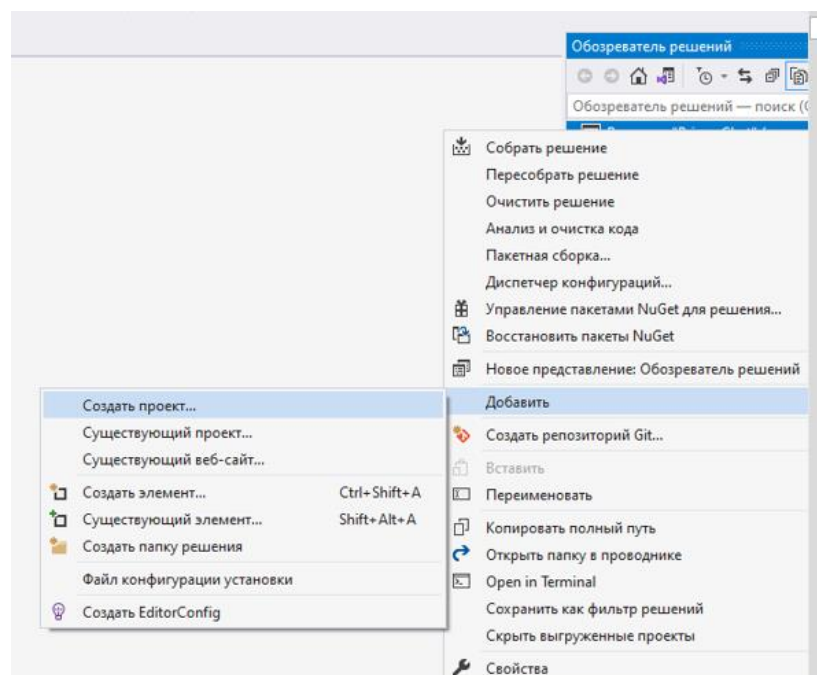
После получения команды от клиента сервер выполняет нужное действие и всем остальным участникам чата (кроме того, кто инициировал событие) высылает информационные сообщения:

- new <имя> - в чате новый участник <имя>;
- message <имя: сообщение> - новое сообщения от участника;
- exit <имя> – пользователь покидает чат.

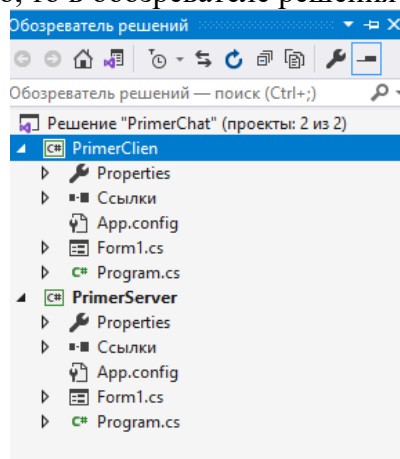
Шаг 1. Открываем Microsoft Visual Studio и создаем 2 новых проекта C#/Windows Forms

В Visual Studio совокупность нескольких проектов называется решением. Сначала создадим решение и проект для сервера, как это делалось в предыдущих лабораторных работах. Например, проекту дадим название PrimerServer, а всему решению имя PrimerChat.

Далее сразу создадим внутри этого решения второй проект PrimerClient. Для этого надо открыть обозреватель решений, правой кнопкой мыши щелкнуть на строке с решением и в меню выбрать команду Добавить/Создать проект.



Если все будет сделано правильно, то в обозревателе решения можно будет увидеть оба проекта.

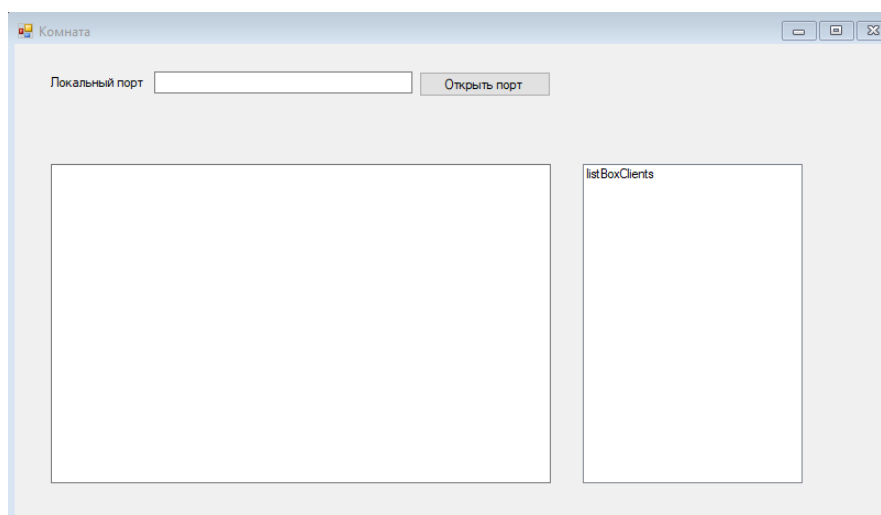


Один из проектов является активным, а другие пассивны. Чтобы сменить активный проект надо в обозревателе решений щелкнуть правой кнопкой на строке с проектом и в меню выбрать пункт «Назначить в качестве запускаемого проекта» (например, PrimerClient)

Шаг 2.1 Создание визуального интерфейса серверного приложения

На рисунке показан визуальный интерфейс, который необходимо создать:

- текстовые поля `textBoxLocalPort` (локальный порт) и `textBoxLog` (поле текстового вывода);
- кнопка `buttonBind` (открыть порт);
- список `listBoxClients` (класс `ListBox`) для отображения списка пользователей;
- таймер `timer1` (класс `Timer`).



Шаг 2.2 Создание исходного кода сервера. Подготовка переменных для хранения данных

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

Серверному приложению понадобится 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.

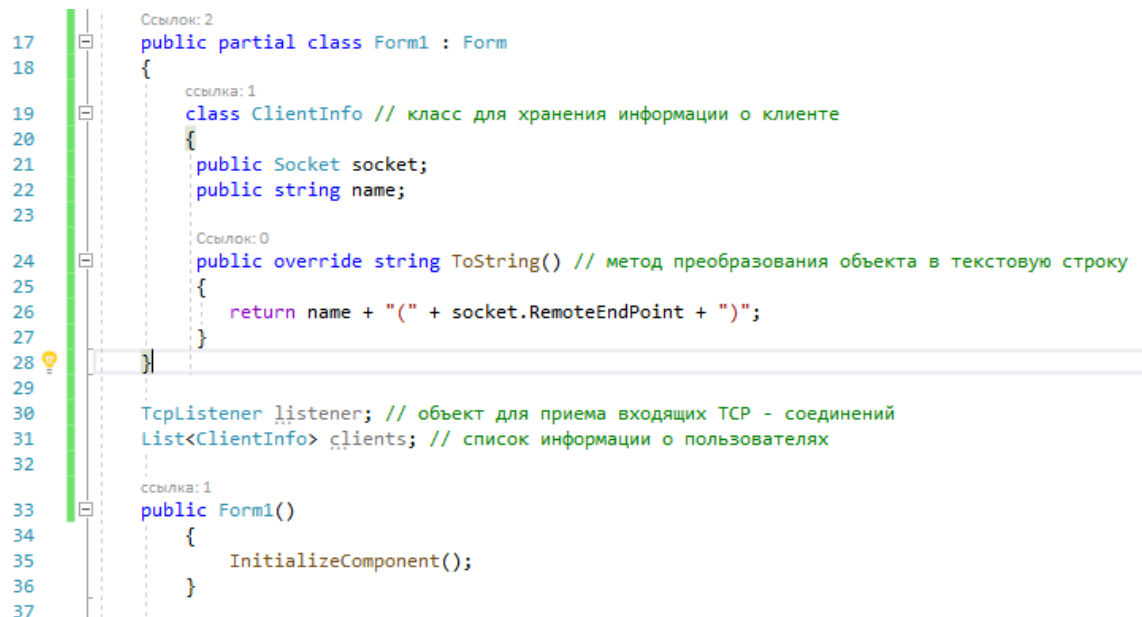
Для этого в программе создадим класс `ClientInfo`.

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>`. Такое объявление гласит, что в переменной `clients` будет храниться список объектов типа `ClientInfo`.



```
17 public partial class Form1 : Form
18 {
19     ссылка: 1
20     class ClientInfo // класс для хранения информации о клиенте
21     {
22         public Socket socket;
23         public string name;
24
25         ссылка: 0
26         public override string ToString() // метод преобразования объекта в текстовую строку
27         {
28             return name + "(" + socket.RemoteEndPoint + ")";
29         }
30     }
31
32     TcpListener listener; // объект для приема входящих TCP - соединений
33     List<ClientInfo> clients; // список информации о пользователях
34
35     ссылка: 1
36     public Form1()
37     {
38         InitializeComponent();
39     }
40 }
```

Шаг 2.3 Создание исходного кода сервера. Создание сокета входящих соединений

Для кнопки `buttonBind` 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 buttonBind_Click(object sender, EventArgs e)
{
    try
    {
        int LocalPort = Int32.Parse(textBoxLocalPort.Text);
        IPEndPoint LocalPoint = new IPEndPoint(IPAddress.Any, LocalPort);
        listener = new TcpListener(LocalPoint);
        listener.Start(); // запускаем процесс прослушивания сокета
        clients = new List<ClientInfo>(); //список пользователей, изначально пустой
        // включаем таймер для периодической проверки
        timer1.Enabled = true;
        // выводим сообщение об успешном открытии сокета
        textBoxLog.AppendText("Открыт TCP порт " + textBoxLocalPort.Text +
            Environment.NewLine);
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}
```

Шаг 2.4 Создание исходного кода сервера. Обработка сообщений от клиента

Для рассылки сообщений клиентами запускается метод `SendToClients`, приведенный ниже. Этот метод высылает команду `command` всем клиентам, но не клиенту, указанному в параметре `exceptOf`.

```
private void SendToClients(string command, ClientInfo exceptOf)
{
    for (int i=0; i<clients.Count; i++)
```

```

    {
        ClientInfo client = clients[i];
        if (client != exceptOf)
        {
            try
            { // преобразуем текст в байты
                byte[] data = Encoding.UTF8.GetBytes(command);
                client.socket.Send(data);
            }
            catch (Exception exc)
            {
                textBoxLog.AppendText(exc.Message + Environment.NewLine);
            }
        }
    }
}

```

Необходимо для кода сервера написать методы CheckListener и DoClient. Коды этих методов можно разместить где угодно внутри класса (но не внутри других методов).

Шаг 2.5 Создание исходного кода сервера. Прием входящих соединений

Ниже приведен исходный код метода CheckListener.

```

private void CheckListener()
{
    if (listener.Pending()) // есть новые соединения
    {
        //надо создать объект для хранения информации о новом пользователе
        //надо создать новый сокет и добавить информацию в список на форме
        ClientInfo newClient = new ClientInfo();
        newClient.socket = listener.AcceptSocket();
        clients.Add(newClient);
        textBoxLog.AppendText("Пользователь " + newClient.socket.RemoteEndPoint + "
подключился"+ Environment.NewLine);
    }
}

```

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.

2.6. Создание исходного кода сервера. Обработка полученного сообщения.

Ниже приведен исходный код метода DoClient, который обрабатывает полученное сообщение.

```

private void DoClient(ClientInfo client, string text_data)
{
    if (text_data.StartsWith("name")) // текст начитается с "name"
    {
        client.name = text_data.Substring(5);
        listBoxClients.Items.Add(client);
        // отправляем пользователям информацию о новом пользователе
        SendToClients("new"+client.name, client);
        textBoxLog.AppendText("Пользователь " + client.socket.RemoteEndPoint + " выбрал
имя " + client.name + "\n");
    }
    if (text_data=="quit") //текст "quit"
    {
        SendToClients("exit" + client.name, client);
    }
}

```

```

        textBoxLog.AppendText("Пользователь " + client.socket.RemoteEndPoint + "
покинул комнату "+ Environment.NewLine);
        client.socket.Shutdown(SocketShutdown.Both);
        client.socket.Close();
        listBoxClients.Items.Remove(client); // убираем из списка окна пользователей
        clients.Remove(client); // убираем из списка пользователей
    }
    if (text_data.StartsWith("message")) // текст начитается с "message"
    {
        string message = text_data.Substring(8); // выделяем из текста сообщение
        SendToClients("message" + client.name + " : " + message, client);
        textBoxLog.AppendText(client.name + " : " + message+ Environment.NewLine);
    }
}

```

Этот метод имеет достаточно большой код, разберем его. Метод состоит из трех секций. Каждая из них проверяет, является ли сообщение одной из команд: name, message или quit. Для проверки используется метод `text_data.StartsWith`, который проверяет, начинается ли строка `text_data` с указанной последовательности символов.

Для команды name 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 `listBoxClients` новый элемент;
- с помощью метода `SendToClients` посылает всем клиентам сообщение new. Для команды quit сервер выполняет следующие действия:
- с помощью метода `SendToClients` посылает всем клиентам сообщение exit;
- разрывает соединение;
- удаляет элемент `client` из поля `listBoxClients` и списка `clients`.

И наконец, для команды message сервер с помощью метода `SendToClients` посылает всем клиентам сообщение message.

2.7. Создание исходного кода сервера. Обработка соединений

Для таймера `timer1` создадим обработчик события `Tick`. Этот обработчик выполняет следующие действия:

проверять наличие новых соединений, вызывая метод `CheckListener`;

в цикле от последнего к первому элементу списка `clients` последовательно проверять сокет на наличие новых данных

если данные имеются, то будет происходить их получение и преобразование в текст;

после этого будет вызываться метод `DoClient`, который обрабатывает полученное сообщение.

```

private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        CheckListener(); // проверяет новые подключения
        for (int i = clients.Count - 1; i >= 0; i--)
        {
            ClientInfo client = clients[i];
            if (client.socket.Available > 0) // если есть новые данные
            {
                // преобразуем текст в байты
                byte[] data = new byte[client.socket.Available];
                int data_size = client.socket.Receive(data);
                string text_data = Encoding.UTF8.GetString(data, 0, data_size);
                // вызываем обработчик DoClient
                DoClient(client, text_data);
            }
        }
    }
    catch (Exception exc)
    {
    }
}

```

```

        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}

```

Написание сервера закончено.

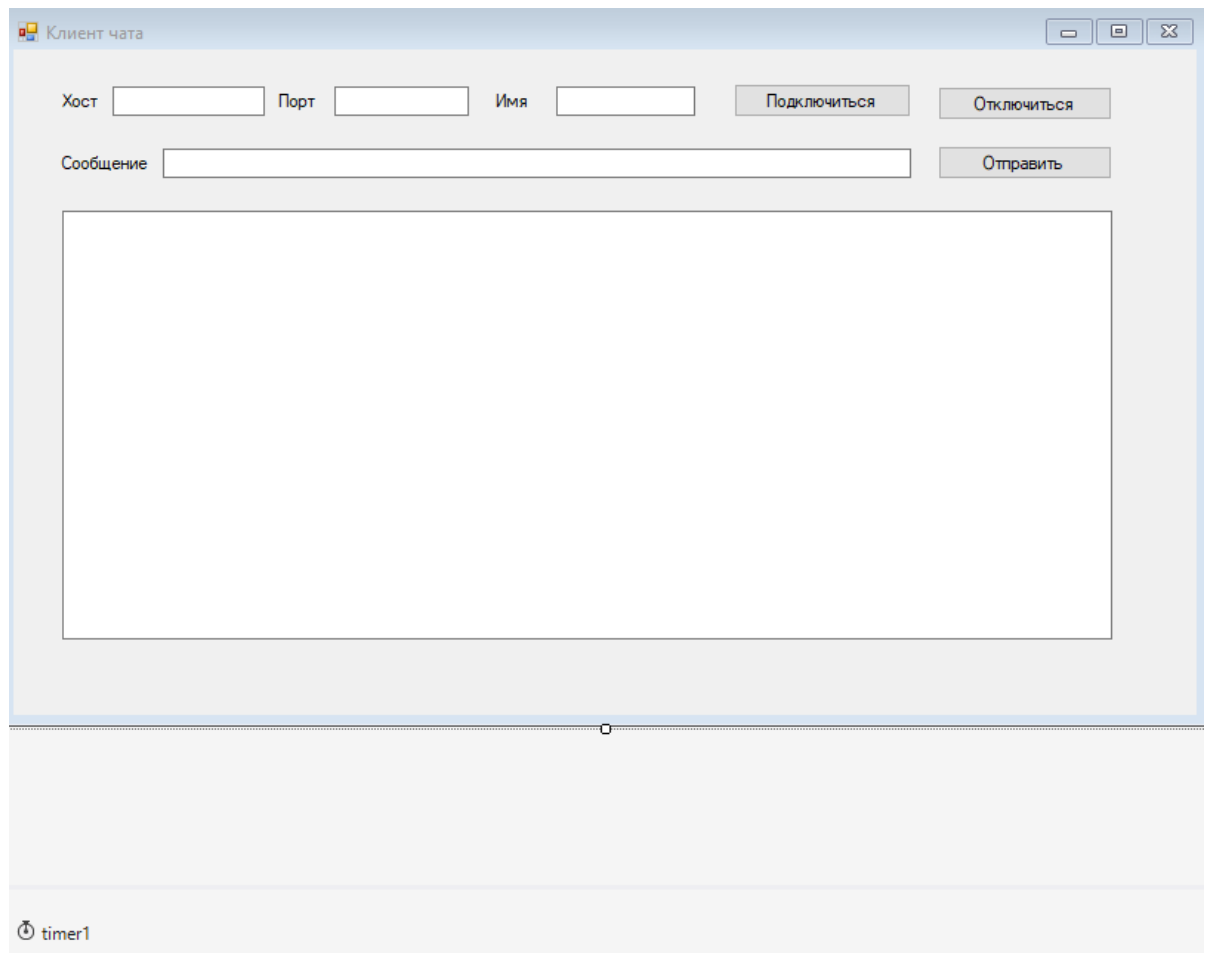
Команда `Environment.NewLine` переводит курсор на следующую строку в объекте `TextBox`.

Шаг 3.1 Создание визуального интерфейса клиентского приложения

Теперь время переключиться на клиентское приложение, которое будет во многом похоже на приложение из лабораторной работы №2.

На рисунке ниже показан визуальный интерфейс, который необходимо создать:

- текстовые поля `textBoxHost` (хост), `textBoxPort` (порт), `textBoxName` (имя), `textBoxMessage` (сообщение) и `textBoxLog` (поле текстового вывода);
- кнопки `buttonConnect` (подключиться), `buttonDisconnect` (отключиться) и `buttonSend` (отправить); для кнопки «Отключиться» задайте свойство `Enabled` (доступно) равным `false`.
- таймер `timer1` (класс `Timer`).



Шаг 3.2 Создание исходного кода клиента

Вначале не забудьте, 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`.

```

Ссылка: 2
public partial class Form1 : Form
{
    Socket socket;

    Ссылка: 0
    private void SendToServer(string command)
    {
        byte[] data = Encoding.UTF8.GetBytes(command); // переводим команду command в список байт
        socket.Send(data); // отправляем данные
    }

    Ссылка: 1
    public Form1()
    {
        InitializeComponent();
    }
}

```

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

- создавать объект Socket, настроенный на протокол TCP;
- подключаться к серверу по данным из textBoxHost и textBoxPort;
- отправлять серверу команду name с данными из textBoxName;
- включить таймер для обработки входящих данных и управлять кнопками Connect/Disconnect.

```

private void buttonConnect_Click(object sender, EventArgs e)
{
    try
    {
        // создаем и сохраняем объект Socket. Параметры: сеть Интернет, данные-поток
        // байт, протокол TCP
        socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
        ProtocolType.Tcp);
        socket.Connect(textBoxHost.Text, Int32.Parse(textBoxPort.Text)); // подключение
        // к серверу
        SendToServer("name" + textBoxName.Text); // передача серверу имени пользователя
        timer1.Enabled = true;
        textBoxLog.AppendText("Подключено к " + textBoxHost.Text + " : " +
        textBoxPort.Text + Environment.NewLine);
        // меняем доступность кнопок "Подключиться"/"Отключиться"
        buttonConnect.Enabled = false;
        buttonDisconnect.Enabled = true;
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}

```

Для кнопки buttonDisconnect создадим обработчик события Click. Этот обработчик выполняет следующие действия:

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

```

private void buttonDisconnect_Click(object sender, EventArgs e)
{
    try
    {
        SendToServer("quit"); //текст "quit", выход из чата
        socket.Shutdown(SocketShutdown.Both); // закрываем получение и отправку данных
        socket.Close(); // закрываем сокет
        timer1.Enabled = false;
    }
}

```

```

        // меняем доступность кнопок "Подключиться"/"Отключиться"
        buttonConnect.Enabled = true;
        buttonDisconnect.Enabled = false;
        textBoxLog.AppendText("Отключено " + Environment.NewLine);
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}

```

Ниже приведен код обработчика кнопки buttonSend. Он просто отправляет серверу команду message.

```

private void buttonSend_Click(object sender, EventArgs e)
{
    try
    {
        SendToServer("message" + textBoxMessage.Text);
        textBoxLog.AppendText(textBoxName.Text + " : " + textBoxMessage.Text +
Environment.NewLine);
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}

```

Последний обработчик – это обработчик таймера timer1. Этот обработчик сначала определяет тип полученного сообщения и в зависимости от этого выводит на экран разную информацию:

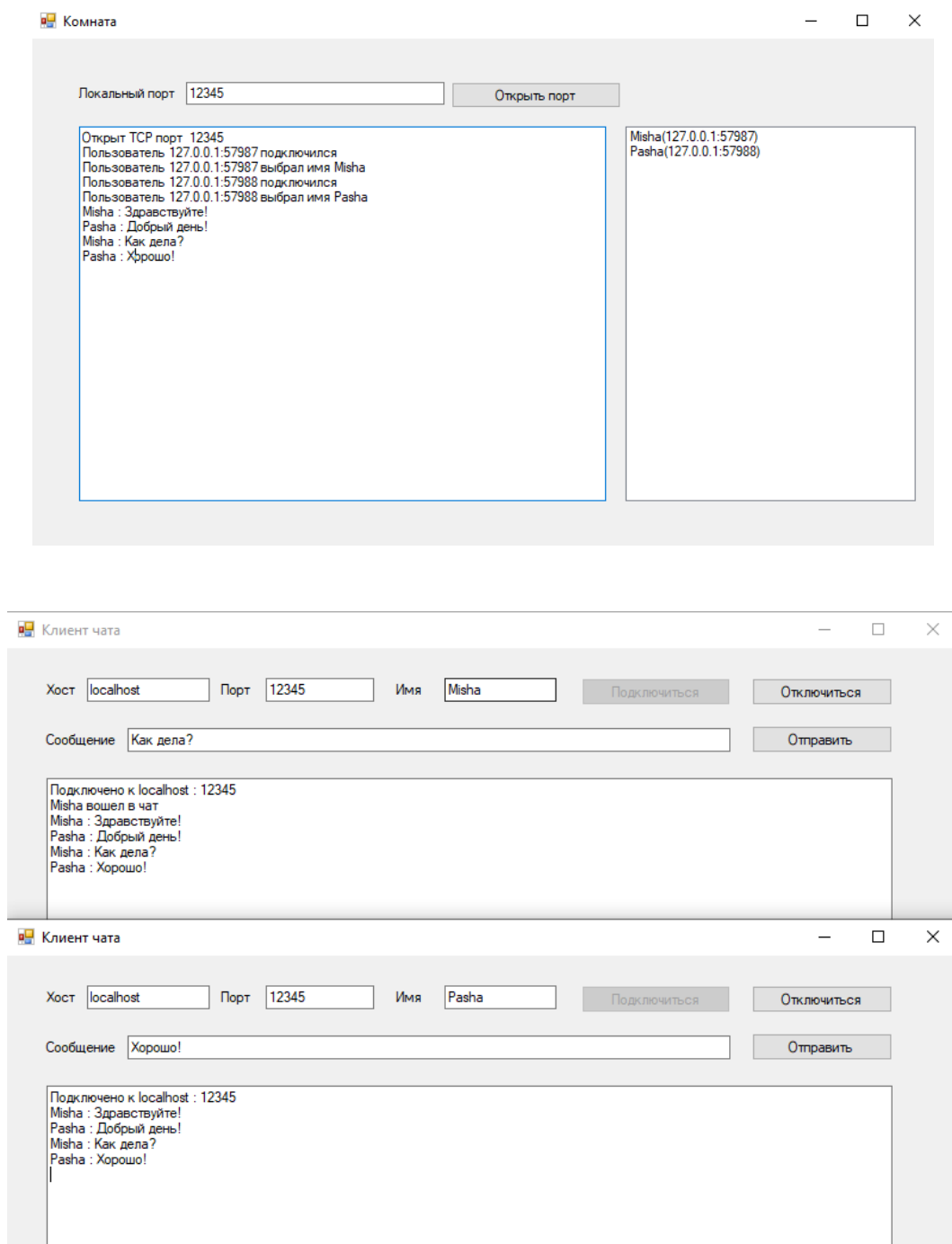
```

private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        if (socket.Available > 0) // если есть новые данные
        {
            // преобразуем текст в байты
            byte[] data = new byte[socket.Available];
            int data_size = socket.Receive(data);
            string text_data = Encoding.UTF8.GetString(data, 0, data_size);
            if (text_data.StartsWith("new")) // текст начинается с "new"
            {
                textBoxLog.AppendText(text_data.Substring(5) + " вошел в чат " +
Environment.NewLine);
            }
            if (text_data.StartsWith("exit")) // текст начитается с "exit"
            {
                textBoxLog.AppendText(text_data.Substring(5) + " покинул чат " +
Environment.NewLine);
            }
            if (text_data.StartsWith("message")) // текст начитается с "message"
            {
                textBoxLog.AppendText(text_data.Substring(8) + Environment.NewLine);
            }
        }
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + Environment.NewLine);
    }
}

```


Шаг 4. Тестирование программы

Сделайте активным проект для сервера и запустите его в режиме без отладки. Откройте порт 12345. Сделайте активным проект клиента и запустите два его экземпляра без отладки. Последовательно подключите оба клиента к серверу. Подключитесь к локальному хосту (localhost), укажите порт 12345 и свое имя в чате (задано ограничение 5 символов). Отправьте сообщения от имени одного клиента другому и наоборот.



Для обоих клиентов разорвите соединение (на скриншотах показан случай, если один из клиентов закрыл комнату не нажав кнопку «Отключиться», второй нажал кнопку «Отключиться»)

