# 1 Задание 1. Основы сокетов на примере .NET



**Цель:** В этом задании мы познакомимся с основами программирования сокетов в .NET Framework с использованием C#. Мы создадим небольшое типовое приложение, состоящее из сервера и клиента, которое будет взаимодействовать по протоколу TCP и UDP.

## 1.1 Критерии оценивания

| № | Задача |
|---|---|
| 1. | Реализуйте TCP Socket сервер и продемонстрируйте соединение с этим сервером с помощью telnet. |
| 2. | Реализуйте клиент TCP Socket и продемонстрируйте соединение между клиентом и сервером. |
| 3. | Внедрение UDP клиента и сервера. Продемонстрируйте соединение и правильную работу клиента и сервера. |

## 1.2 Методические рекомендации

### 1.1.1 Основы работы с сетью:

Межпроцессная коммуникация, т.е. возможность двух или более физически подключенных машин обмениваться данными, играет очень важную роль в разработке корпоративного программного обеспечения. TCP/IP является наиболее распространенным стандартом, принятым для такой связи. В рамках TCP/IP каждая машина идентифицируется уникальным 4-байтовым целым числом, называемым ее IP-адресом (обычно отформатированным как 192.168.0.101). Для удобства запоминания этот IP-адрес в основном привязан к удобному для пользователя имени хоста. Программа, приведенная ниже (*showip.cs*), использует класс `System.Net.Dns` для отображения IP-адреса машины, имя которой передается в первом аргументе командной строки. При отсутствии аргументов командной строки она отображает имя и IP-адрес локальной машины.

```
using System;
using System.Net;
class ShowIP{
    public static void Main(string[] args){
        string name = (args.Length < 1) ? Dns.GetHostName() : args[0];
        try{
            IPAddress[] addrs = Dns.Resolve(name).AddressList;
```

```
            foreach(IPAddress addr in addrs)
                Console.WriteLine("{0}/{1}",name,addr);
        }catch(Exception e){
            Console.WriteLine(e.Message);
        }
    }
}
```

`Dns.GetHostName()` возвращает имя локальной машины, а `Dns.Resolve()` возвращает `IPHostEntry` для машины с заданным именем, свойство `AddressList` которого возвращает IP-Аадреса машины. Метод `Resolve` вызовет исключение, если указанный хост не найден.

Хотя IP-адрес позволяет идентифицировать машины в сети, каждая машина может содержать несколько приложений, которые используют сеть для обмена данными. Согласно TCP/IP, каждое сетевое приложение привязывает себя к уникальному 2-байтному целому числу, называемому его **номером порта**, которое идентифицирует это приложение на машине, на которой оно выполняется. Передача данных происходит в виде байтовых пакетов, называемых IP-пакетами или датаграммами. Размер каждой датаграммы составляет 64 Кбайта и содержит данные для передачи, фактический размер данных, IP-адреса и номера портов отправителя и предполагаемого получателя. Как только датаграмма помещена в сеть машиной, она будет физически принята всеми остальными машинами, но будет обработана только той машиной, IP-адрес которой совпадает с IP-адресом получателя в пакете. Позже, эта машина передаст пакет запущенному на ней приложению, которое привязано к номеру порта получателя, присутствующему в пакете.

Пакет TCP/IP на самом деле предлагает два различных протокола для обмена данными. Протокол управления передачей (*Transmission Control Protocol* - TCP) является надежным протоколом, ориентированным на работу в рамках установленного соединения, в то время как *User Datagram Protocol* - UDP не очень надежный (но быстрый) протокол без соединения.

### 1.1.2 Разработка сервера TCP-сокета

В TCP существует четкое различие между серверным и клиентским процессами. Процесс сервера начинается на хорошо известном порту (о котором известно клиентам) и прослушивает входящие запросы на соединение. Клиентский процесс запускается на любом порту и выдает запрос на соединение.

Основные шаги по созданию TCP/IP сервера следующие:

1. Создайте `System.Net.Sockets.TcpListener`, передайте ему локальный адрес и порт:
```
TcpListener listener = new TcpListener(local_port);
listener.Start();
```
2. Дождитесь входящего запроса на подключение и примите объект `System.Net.Sockets.Socket` от слушателя всякий раз, когда появляется запрос:
```
Socket soc = listener.AcceptSocket(); // blocks
```
3. Создайте `System.Net.Sockets.NetworkStream` из `Socket`:
```
Stream s = new NetworkStream(soc);
```
4. Организуйте коммуникацию с клиентом, используя предопределенный протокол:
5. Закройте `Stream`:
```
s.Close();
```
6. Закройте `Socket`:

```
s.Close();
```

7. Идите к шагу **2**.

Обратите внимание, что когда один запрос принимается на шаге 2, никакой другой запрос не будет принят до тех пор, пока код не достигнет этапа 7. (Запросы будут помещены в очередь или в бэклог.) Для того чтобы принять и обслужить одновременно более одного клиента, шаги 2 - 7 должны быть выполнены в несколько потоков. Программа ниже (emptcpserver.cs) является многопоточным TCP/IP сервером, который принимает имя сотрудника от своего клиента и отсылает обратно задание сотрудника. Клиент завершает сессию, посылая пустую строку вместо имени сотрудника. Данные сотрудника извлекаются из словаря, определенного в начале определения класса EmployeeTCPServer.

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

class EmployeeTCPServer
{
    static TcpListener listener;

    const int LIMIT = 5; //можем обработать сразу 5 клиентов одновременно

    //как-бы база данных
    //словарь, где хранится вся информация о сотрудниках
    static Dictionary<string, string> employees =
        new Dictionary<string, string>()
    {
        {"john", "manager"},
        {"jane", "steno"},
        {"jim", "clerk"},
        {"jack", "salesman"}
    };

    public static void Main()
    {
        //Порт нашего сервера
        Int32 port = 13000;
        //IP-адрес нашего сервера – локальная машина
        IPAddress localAddr = IPAddress.Parse("127.0.0.1");

        listener = new TcpListener(localAddr, port);
        listener.Start();
#if LOG
            Console.WriteLine("Server mounted,
                            listening to port 13000");
#endif
        //We would launch our server in a separate thread
        for (int i = 0; i < LIMIT; i++)
        {
            Thread t = new Thread(new ThreadStart(Service));
            t.Start();
        }
    }
    public static void Service()
    {
        //постоянно ждем входящего соединения
        while (true)
        {
            Socket soc = listener.AcceptSocket();
```

```
#if LOG
                Console.WriteLine("Connected: {0}",
                                        soc.RemoteEndPoint);
#endif
            try
            {
                Stream s = new NetworkStream(soc);
                StreamReader sr = new StreamReader(s);
                StreamWriter sw = new StreamWriter(s);
                sw.AutoFlush = true; // enable automatic flushing

                sw.WriteLine("{0} Employees available",
                      employees.Count);

                while (true)
                {
                    string name = sr.ReadLine();
                    if (name == "" || name == null) break;
                    string job =
                        employees[name];
                    if (job == null) job = "No such employee";
                    sw.WriteLine(job);
                }
                s.Close();
            }
            catch (Exception e)
            {
#if LOG
                    Console.WriteLine(e.Message);
#endif
            }
#if LOG
                Console.WriteLine("Disconnected: {0}",
                                        soc.RemoteEndPoint);
#endif
            soc.Close();
        }
    }
}
```

Код между `#if LOG` и `#endif` будет добавлен компилятором только в том случае, если при компиляции определен символ LOG (условная компиляция). Скомпилировать указанную программу можно либо определив символ LOG (информация заносится в журнал на экране):

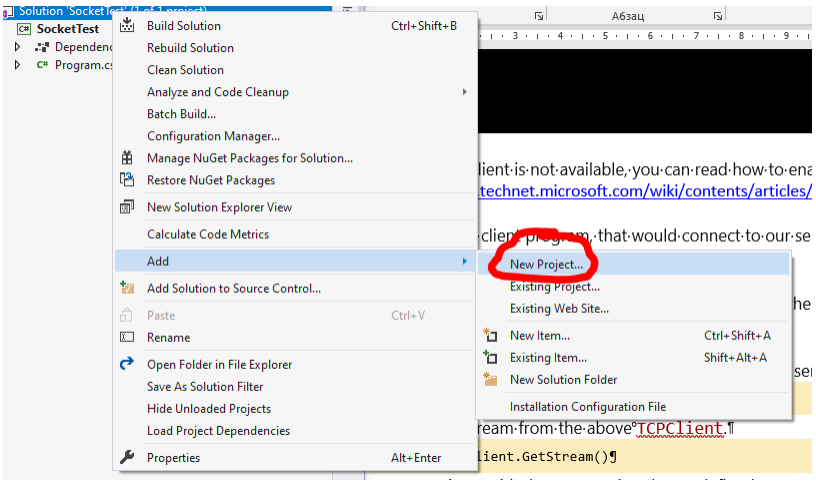
Для тестирования сервера, вы можете использовать: `telnet localhost 13000`.

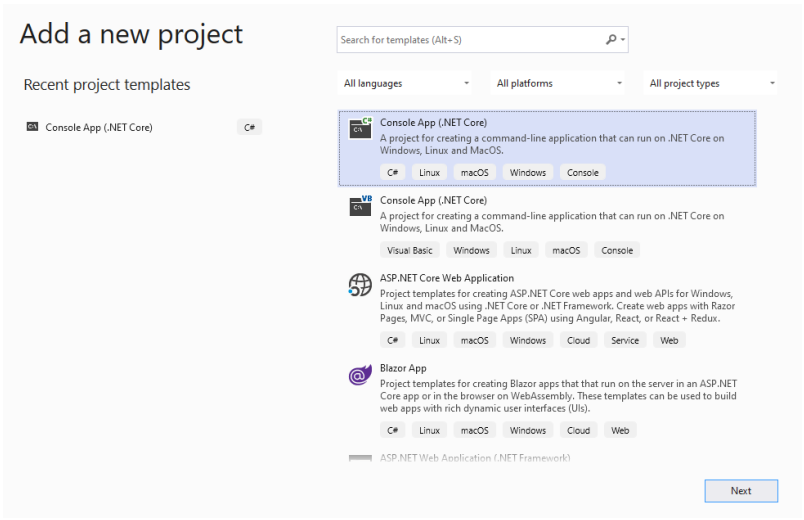

Если telnet-клиент недоступен, вы можете прочитать, как его включить здесь:
https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx

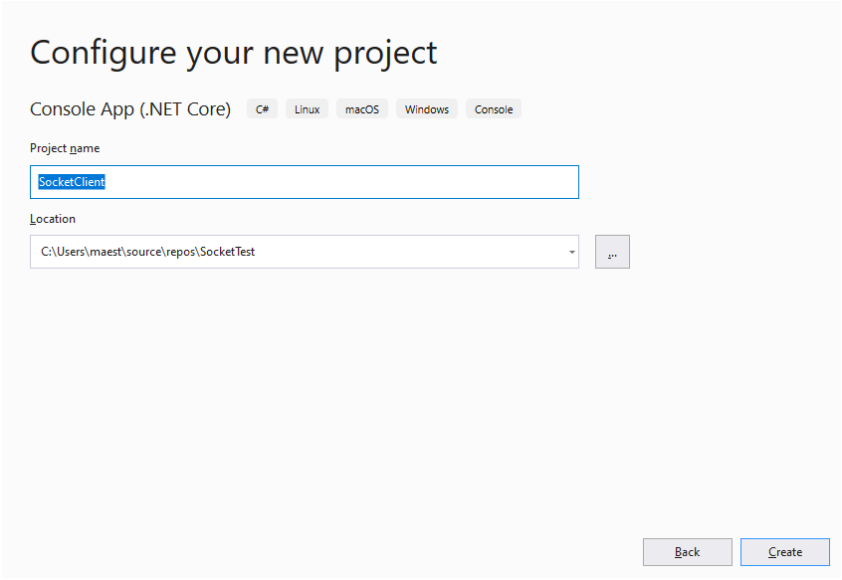
### 1.1.3 Разработка клиента TCP

Давайте создадим клиентскую программу, которая будет подключаться к нашему серверу. Основные шаги по созданию TCP/IP клиента следующие:

0. В текущем решении создайте новый проект с именем SocketClient



1. Создайте `System.Net.Sockets.TcpClient` передав адрес и порт сервера:

```
TcpClient client = new TcpClient(host, port);
```

2. Получите stream из `TCPClient`.

```
Stream s = client.GetStream()
```

3. Организуйте общение с сервером по соответствующему протоколу
4. Закройте `Stream`:

```
s.Close();
```

5. Закройте соединение:

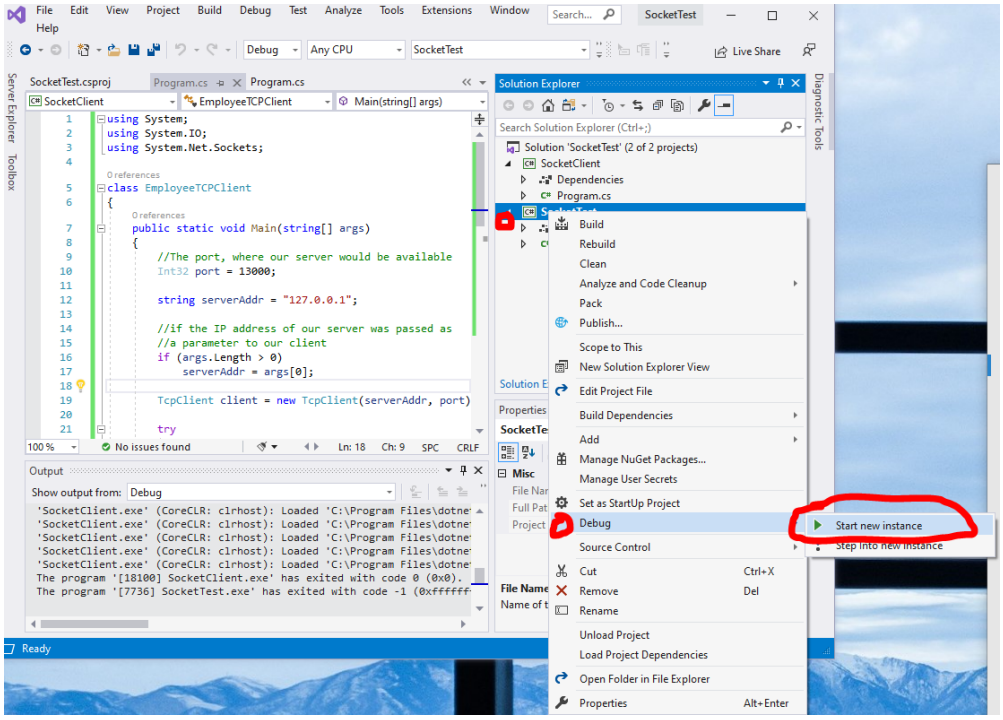
```
client.Close();
```

Программа ниже (*emptcpclient.cs*) обеспечивает общение с `EmployeeTCPServer`:

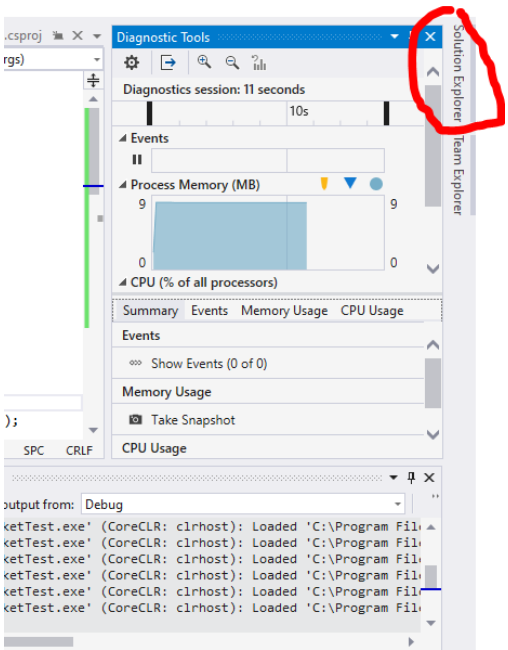
```
using System;
using System.IO;
using System.Net.Sockets;

class EmployeeTCPClient
{
    public static void Main(string[] args)
    {
        //Порт нашего сервера
        Int32 port = 12000;

        string serverAddr = "127.0.0.1";

        //Если адрес сервера передан как параметр
        if (args.Length > 0)
            serverAddr = args[0];

        TcpClient client = new TcpClient(serverAddr, port);

        try
            {
                Stream s = client.GetStream();
                StreamReader sr = new StreamReader(s);
                StreamWriter sw = new StreamWriter(s);
                sw.AutoFlush = true;
                Console.WriteLine(sr.ReadLine());
                while (true)
                {
                    Console.Write("Name: ");
                    string name = Console.ReadLine();
                    sw.WriteLine(name);
                    if (name == "") break;
                    Console.WriteLine(sr.ReadLine());
                }
                s.Close();
            }
            finally
            {
                // code in finally block is guaranteed
                // to execute irrespective of
                // whether any exception occurs or does
                // not occur in the try block
                client.Close();
            }
    }
}
```

Чтобы протестировать клиентское приложение, необходимо запустить сервер, а затем запустить клиента, чтобы он смог подключиться к рабочему серверу. Сделать это можно из среды Visual Studio вот так.

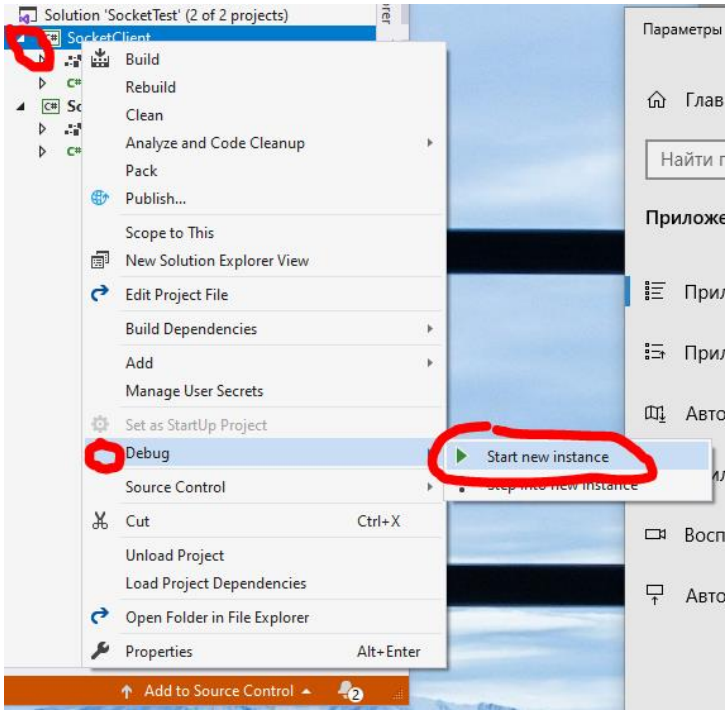
1. Щелкните правой кнопкой мыши на проекте сервера в solution explorer в правой части окна VS. В моем случае он имеет название "SocketTest". Выберите “Debug”->”Start new instance”. Это позволит запустить новый экземпляр сервера.




2. После запуска сервера вы можете запустить клиента. Перейдите на вкладку " Solution explorer " в VS




3. Кликните правой кнопкой мыши на проекте клиента в проводнике решений справа от окна VS. В моем случае он имеет название "SocketClient". Выберите " Debug"->" Start new instance" (Отладка)-> "Start new instance" (Запуск новой копии). Это позволит запустить новый экземпляр клиента.

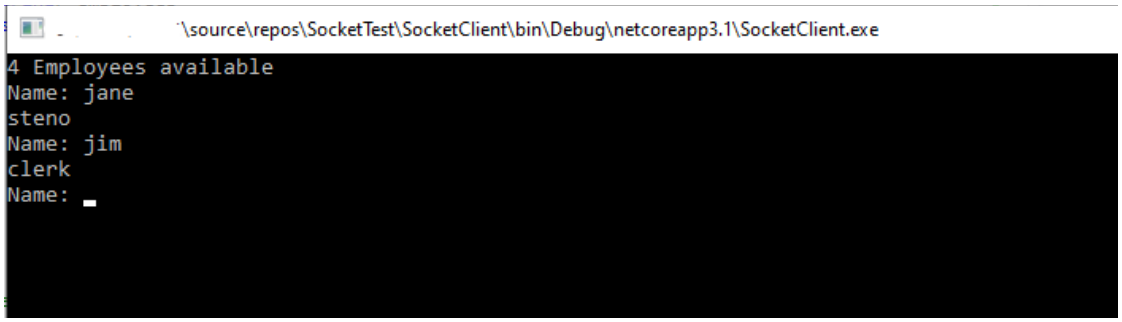



4. Упс, я думаю, мы получили исключение: соединение было отклонено узлом сервера. Как это может быть? Конечно! В коде клиента мы ошиблись номером порта. Остановите выполнение клиента и сервера, перейдите к исходному коду клиента и измените строку

```
Int32 port = 12000;
```
на

```
Int32 port = 13000;
```

5. Сохраните проект и попробуйте запустить его еще раз: сервер - сначала, затем - клиент. Теперь он должен работать.




### 1.1.4 Разработка UDP-клиента и сервера.

В отличие от TCP, UDP не использует соединения, т.е. данные могут быть отправлены на несколько приемников с помощью одного сокета

1. Создайте `System.Net.Sockets.UdpClient` используя локальный или удаленный адрес и порт:

```
UdpClient client = new UdpClient(local_ port);
```

или

```
UdpClient client = new UdpClient(remote_host, remote_port);
```

2. Получите данные с помощью `UdpClient`:

```
System.Net.IPEndPoint ep = null;
byte[] data = client.Receive(ref ep);
```

`byte` массив `data` будет содержать полученные данные, а `ep` будет содержать адрес отправителя.

3. Отправьте данные используя `UdpClient`.

Если имя удаленного хоста и номер порта уже были переданы `UdpClient` через его конструктор, то отправьте `data byte` массива, используя:

```
client.Send(data, data.Length);
```

В противном случае, отправьте данные используя `IPEndPoint ep` получателя:

```
client.Send(data, data.Length, ep);
```

Программа ниже (empudpserver.cs) получает имя сотрудника от удаленного клиента и отправляет обратно его должность, используя UDP:

```
using System;
using System.Net;
using System.Text;
using System.Net.Sockets;
using System.Collections.Generic;

class EmployeeUDPServer
{
    //our "poor man's" database emulator
    //a dictionary where we would store all inromation
    static Dictionary<string, string> employees =
        new Dictionary<string, string>()
    {
        {"john", "manager"},
        {"jane", "steno"},
        {"jim", "clerk"},
        {"jack", "salesman"}
    };
    public static void Main()
    {
        Int32 port = 13000;
        UdpClient udpc = new UdpClient(port);
        Console.WriteLine("Server started, servicing on port "+port.ToString());
        IPEndPoint ep = null;
        while (true)
        {
            byte[] rdata = udpc.Receive(ref ep);
            string name = Encoding.ASCII.GetString(rdata);
            string job = employees[name];
            if (job == null) job = "No such employee";
            byte[] sdata = Encoding.ASCII.GetBytes(job);
            udpc.Send(sdata, sdata.Length, ep);
        }
    }
}
```

Следующая программа (empudpclient.cs) является UDP клиентом вышеуказанной серверной программы:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class EmployeeUDPClient
{
    public static void Main(string[] args)
    {
        string ipadress = "127.0.0.1";
        Int32 port = 13000;
        if (args.Length > 0)
            ipadress = args[0];

        UdpClient udpc = new UdpClient(ipadress, port);
        IPEndPoint ep = null;
        while (true)
        {
            Console.Write("Name: ");
            string name = Console.ReadLine();
            if (name == "") break;
            byte[] sdata = Encoding.ASCII.GetBytes(name);
            udpc.Send(sdata, sdata.Length);
            byte[] rdata = udpc.Receive(ref ep);
            string job = Encoding.ASCII.GetString(rdata);
            Console.WriteLine(job);
        }
    }
}
```

Источник: http://www.codeproject.com/Articles/10649/An-Introduction-to-Socket-Programming-in-NET-using