NETWORK PROGRAMMING

NETWORK PROGRAMMING
IN .NET FRAMEWORK

TCP AND UDP SOCKETS, UNICAST, BROADCAST, MULTICAST

USING NETWORK PROTOCOLS HTTP, SMTP, FTP

Lesson 1, 2

Introduction to networks, sockets

Asynchronous sockets

Contents

1. What is Network Programming?

In this lesson we start to study a new training course «network programming». As we can see from the course title, all technologies we are going to study, are directly related to development of the applications designated for the networking interaction between several computers.

Starting from the 60-s of the XXth century was launched the process of creating Area Networks. On the 29th of October 1969 at 21:00 there was the first public demonstration of the interaction possibility of several computers by means of telephone networks. The net comprised two terminals, one of which was at California University and the second one situated at the distance of 600 km from it — at Stanford University.

The development was coordinated by the supervision of Defense Advanced Research Project Agency, USA (DARPA). And error tolerance was one of the most essential peculiarity of the newly created network, meaning that even upon the interaction at a network part (for example due to a nuclear attack over the territory of the country) the rest of the network could continue functioning.

Thus, Local Area Networks have been widely used since the 70-s of the XX century. Consequently, there arouse a necessity to develop the software providing interaction of software products.

Therefore, at the time a network couldn't have easily interacted with the other networks built on the basis of other technical standards. The end of the 1970-s was the beginning of the development of data communication protocols that were standardized by the mid 80-s.

Before the 90-s of the XXth century a network was mainly used for sending mails, it was the time when the mailing lists, news groups and message boards appeared.

Since the 1st of January 1983 ARPA network has been used for TCP/IP protocol operation instead of NCP. Thus far, a TCP/IP is successfully used for network integration (also they say «lay up»). Exactly during the time (in 1983) ARPA network got fixed with a name «Internet».

2. Objectives and Tasks of Network Programming

In view of such a turbulent development of data communication protocols there appeared a new application class that cannot exist without the network. Such applications received a name of ***client-server*** applications.

While working within a network one computer acts as a ***server*** and the second one connects it as a ***client***, and it is implied that both computers use one connection method (for example, cable type, signal swing, modulation system) and use the same language for communication.

In order to standardize exchange methods between the computers there should be used special set of data exchange rules that are also called data communication ***protocol***.

Upon this, the task is split into two parts minimum: part of the application logic is realized within a server, and another part of the logic — within a client. Thus, most often there is no user interface provided within a server side. Moreover, client side is usually amounts to user interface with network tools of interaction with a server.

Client-server applications allow you to distribute computing tasks between multiple computers within a network that optimizes the tasks performed on several machines.

Client-server applications also allow splitting the tasks into the parts, assigning the solving of the task to several computers within a network, this technology was called a technology of ***distributed computing***.

Therefore, we can draw out the main objective of network applications developments and it is creation of effective and safe interaction of different computers either within LANs (intranet), or scattered throughout the world (Internet).

3. What is a Network?

Let us define the terms that we will review in the present lesson. So, the first thing we have to do it to understand what a «network» is.

Network is a group of the interconnected computers and devices by means of communication links.

All these devices (computers, routers, gateways, printers) are called nodes. Nodes are interconnected by means of the channels, and any communication links can be used in their capacity (cable, optical cable, optical atmospheric, radio, etc.). Network nodes can communicate with each other within these ***communication links*** sending messages to each other.

According to the size, networks are classified into ***LANs,*** uniting the nodes in the framework of one building or within a group of closely situated buildings, and ***WANs***, which unite several LANs.

4. OSI Model

In order to formalize data exchange protocols through communication links an ISO organization adopted a standard protocol model that was called a 7-layer OSI model (Open System Interconnection). Main tasks of network nodes interaction were drawn out in the framework of this model.

Each layer of OSI model has its own assignation and each of them is connected to an above and below layer.

Here are these 7 layers:

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

A problem of two computer interaction within a network can be resolved only in case if a developer who is programming this interaction, clearly understands network operation.

You have just finished the course «Introduction into networking technologies», where you have studied an OSI model, building and functioning principles of LANs and WANs in details.

Let me remind you in short about the purpose of one or another OSI model layer and I will show you the moments that are of the initial interest to a developer.

- ***On a physical layer*** we define the data transmission medium, transferred information encoding methods, slots and cables applied for network construction.
  A developer, dealing with the resolving of applied tasks, has nothing to do at the layer.
- ***Data link layer.*** At first it tests the availability of transmission medium. Second, it realizes errors detection and correction mechanisms.
- ***Network layer*** implies the formation of an integrated transport system uniting several networks. However, these networks can use totally different communication principles and have a randomly organized structure!

5. Basic Terms

At this level we use a nodes logic addressing (unlike physical addresses of a data link layer). The fact is that physical addresses of a data link layer can be used within LAN only.

Usually (or commonly) an IP protocol is used at the network layer, meaning that each network node is assigned with a unique network IP address.

While realizing a network application the developer should know an IP address of the node he wants to connect to and IP address of the node that he will be connected to.

At the network layer a developer creates a special network API object — socket (slot), which connects a particular network address and transport type protocol.

Using a socket the developer can realize interaction with the other socket within a network.

Next layer is a ***transport*** layer. And we will have to stop in more details on the topic, because exactly at a transport layer we face applications for the first time. At the transport layer we also face a notion of ***a port*** or end point, which describes an application that is ready to accept a network connection or connect with an application situated on the other node. At the transport layer the developer should choose which of the protocols he is going to use. There are several popular protocols of the transport layer. The most popular protocols among the developers are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), RTP protocol (Real-time Transport Protocol) is not so often used, and others.

TCP protocol is used when we need a guarantee of packages delivery to a remote node. Working through the TCP protocol a client establishes a connection with a server at the beginning of a session, and by the end thereof he breaks up the connection. Data transmission is possible only if there is a connection. In this respect a receiving side transmits to a sending side a conformation of data accept, in case of distortion of transmitted data or its loss, the sending side repeats package sending.

Within ***UDP*** protocol data transmission is exercised without delivery guarantee and without maintaining a permanent connection. Maintaining data transmission integrity should be realized by upper layers protocols or to be ignored.

UDP protocol is often used in the cases where transmission rate counts more and it is possible to neglect some part of the data.

If it is important to preserve the sequence of transmitted frames, then it would make sense to use RTP protocol, which is usually applied upon transfer of multimedia data (voice, video). RTP protocol is based on the UDP protocol and differs only by the fact that there is information in the contents of each package, allowing restoring the packages sequence valid upon the transmission.

Session layer defines an order of connection establishment, i.e. an order session initialization, an order of its exercising. ***Realization of this layer is a responsibility of a developer***.

Exactly at this layer we define an order of messages exchange between a client socket and server socket.

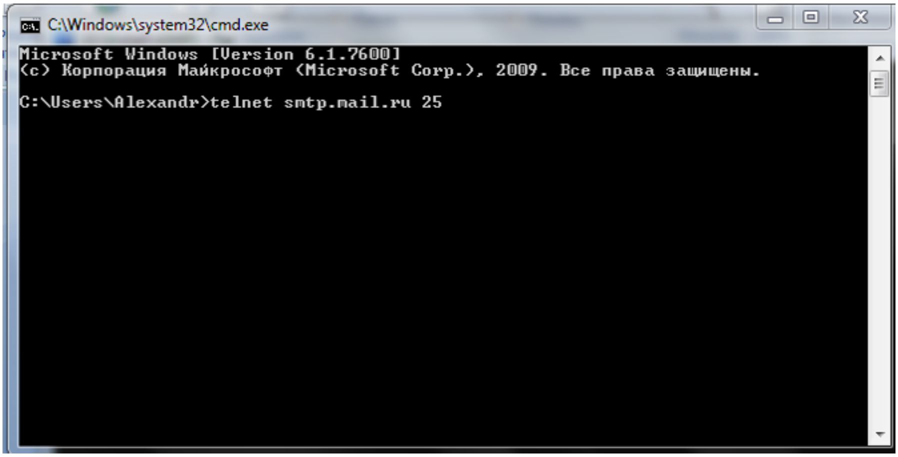
At the ***presentation*** layer we define a form of transmitted message. Here a developer chooses a presentation of his message, i.e. the form in which he will transmit and receive data, either in the binary sequence or in a form of the commands in a text mode (ASCII or UNICODE), it also can make sense to transmit data into XML, or to serialize them within a Soap? Whether to apply encoding or not? These issues of the presentation layer should be resolved during the development of own protocol stack of network exchange.

Application is the uppermost layer of an ***OSI model***. This layer realizes a functionality of the upper layer, such as mail transmission (SMTP), receiving mails from the remote server (POP), files transfer via network architecture (FTP and TFTP), preview of the web-pages (HTTP and HTTPS), transfer of the voice via VoIP (SIP) and other standard protocols. A developer should strictly follow the relative protocol while developing his own applications for resolving trivial tasks.

If during an operation arises a necessity (and it arises quite often) to develop an own application layer protocol, then it is necessary to elaborate a system of the commands to a server of own protocol and system of server's requests.

So, here are all eight layers of an ***OSI model***. As you have already noticed that software and hardware part of network structure are separated within a model. First two layers — are the layers, which operate with the hardware features of a network, they depend on the top and network equipment. The rest five upper layers little depend in technical peculiarities of network construction. You can shift to another networking technology, and it won't require any changes within the software features of the upper layers.

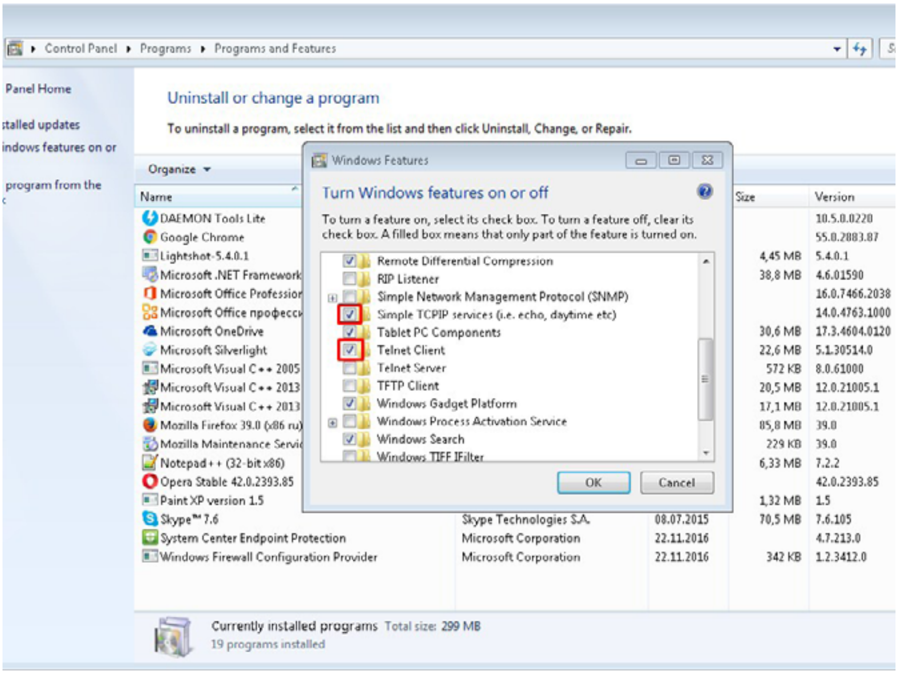
Here is a simple example of sending a mail message via the SMTP server. This can be done using the telnet program — it is a very useful utility for establishing a TCP connection (transport layer) with a remote node (network layer), providing the commands transfer (application layer) in a form of ASCII string (presentation layer). Telnet client initiates a connection (session layer) with the SMTP server using the following commands: telnet smtp.mail.ru:



```
Microsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\Alexandr>telnet smtp.mail.ru 25
```

smtp.mail.ru — node name, and 25 — is a standard TCP port of the SMTP server.

By the way, the owners of Windows 7 pro should install a Telnet client through the snap-in «software and components» -> «enabling and disabling windows components» of the control panel, Telnet will prove useful at the stage of networking applications testing; moreover, you should enable simple services TCP/IP, for we will need them while testing simple client applications:



Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ✓ Remote Differential Compression
- ☐ RIP Listener
- ☐ Simple Network Management Protocol (SNMP)
- ✓ Simple TCPIP services (i.e. echo, daytime etc)
- ✓ Tablet PC Components
- ✓ Telnet Client
- ☐ Telnet Server
- ☐ TFTP Client
- ✓ Windows Gadget Platform
- ☐ Windows Process Activation Service
- ✓ Windows Search
- ☐ Windows TIFF IFilter

Name			Size	Version
DAEMON Tools Lite				10.5.0.0220
Google Chrome				55.0.2883.87
Lightshot-5.4.0.1			4,45 MB	5.4.0.1
Microsoft .NET Framework			38,8 MB	4.6.01590
Microsoft Office Profession				16.0.7466.2038
Microsoft Office професс				14.0.4763.1000
Microsoft OneDrive			30,6 MB	17.3.4604.0120
Microsoft Silverlight			22,6 MB	5.1.30514.0
Microsoft Visual C++ 2005			572 KB	8.0.61000
Microsoft Visual C++ 2013			20,5 MB	12.0.21005.1
Microsoft Visual C++ 2013			17,1 MB	12.0.21005.1
Mozilla Firefox 39.0 (x86 ru)			85,8 MB	39.0
Mozilla Maintenance Servic			229 KB	39.0
Notepad++ (32-bit x86)			6,33 MB	7.2.2
Opera Stable 42.0.2393.85				42.0.2393.85
Paint XP version 1.5			1,32 MB	1.5
Skype™ 7.6	Skype Technologies S.A.	08.07.2015	70,5 MB	7.6.105
System Center Endpoint Protection	Microsoft Corporation	22.11.2016		4.7.213.0
Windows Firewall Configuration Provider	Microsoft Corporation	22.11.2016	342 KB	1.2.3412.0

Currently installed programs Total size: 299 MB
19 programs installed

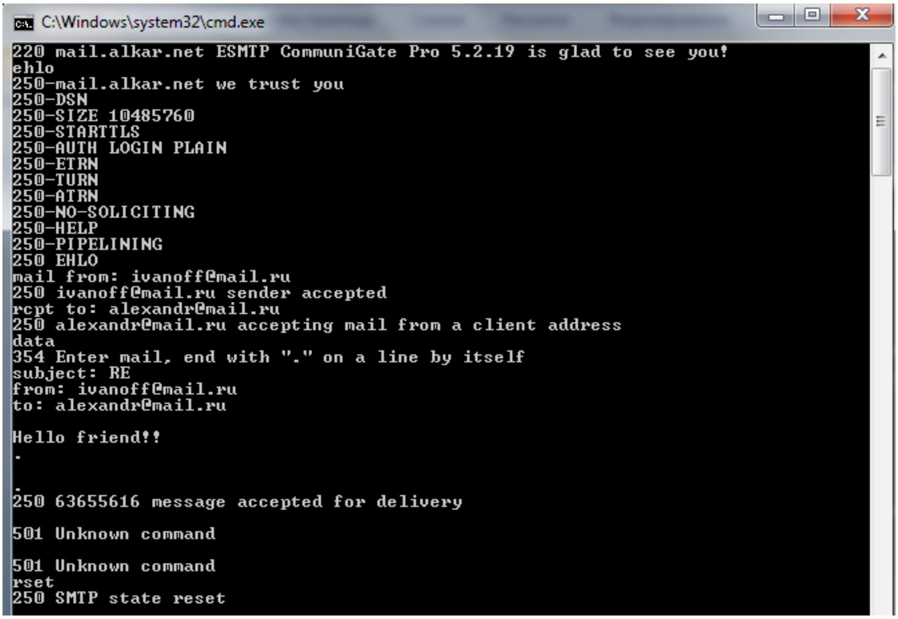
After establishing the connection with the smtp server it sends a client a string containing the code 220 (readiness) and its information. Client sends a server the greeting «ehlo» (meaning that the server is the first to send it to a client, and the client responses to a server — a communication order is defined by a session layer).

Data exchange is exercised in a form of the strings in ASCII 7-bit encoding, which is defined by a presentation layer by the SMTP protocol.

Commands sent to a server and received responses are defined by an application layer and described within a relevant document RFC-821 (RFC-2821 for ESMTP).

You can get familiar with the relevant documents at the site http://ietf.org.

Generally speaking it looks like that:



C:\Windows\system32\cmd.exe

```
220 mail.alkar.net ESMTP CommuniGate Pro 5.2.19 is glad to see you!
ehlo
250-mail.alkar.net we trust you
250-DSN
250-SIZE 10485760
250-STARTTLS
250-AUTH LOGIN PLAIN
250-ETRN
250-TURN
250-ATRN
250-NO-SOLICITING
250-HELP
250-PIPELINING
250 EHLO
mail from: ivanoff@mail.ru
250 ivanoff@mail.ru sender accepted
rcpt to: alexandr@mail.ru
250 alexandr@mail.ru accepting mail from a client address
data
354 Enter mail, end with "." on a line by itself
subject: RE
from: ivanoff@mail.ru
to: alexandr@mail.ru

Hello friend!!
.

.
250 63655616 message accepted for delivery

501 Unknown command

501 Unknown command
rset
250 SMTP state reset
```

To be honest I haven't sent a mail in the end, because an addressee, which address I just made up, would be quite surprised to get a letter from an unknown addressee. In addition, the smtp.mail.ru server also requires authentication procedures, which is absent in the above mentioned fragment. And don't forget that a mail heading will contain a sender's IP address.

Possibility to interact with the other systems within a network in the operating systems of Windows NT family is provided by means of the sockets — compatible and almost complete analogue of Berkeley Sockets library — developed in 1983 for a simpler realization of network interaction in operating systems of UNIX family.

The very term SOCKET means a nest or a plug (the term stood for the nests at the telephone offices with manual switching).

Within the operating systems, starting from the Windows NT, the second version of Windows Sockets library is used. This library is totally available for a developer on WinAPI, but it contains an unmanaged code. It was natural that developers of the .Net Framework have created a managed library, which encapsulate platform invokes providing a developer with a handy tool for writing networking applications.

Though, this tool is completely compatible with an unmanaged library WinSock2 and with Berkeley Sockets UNIX systems, it allows building distributed systems where the nodes may function on different platforms.

6. Socket Class

Classes for working with the network are situated within name spaces System.Net and System.Net.Sockets;

System.Net.Sockets.Socket Class

Thus, Socket class is a class that realizes socket interface Berkeley on a Microsoft.Net Framework platform. As it is shown in MSDN, Socket has a set of methods and features for realization of network interaction. Socket class allows transmitting and accepting data using any of communication protocols that are within s ProtocolType list, such as:

Member name	Description
IP	IP protocol.
Icmp	ICMP protocol.
Igmp	IGMP protocol.
Ggp	GGP protocol.
IPv4	IPv4 protocol.
Tcp	TCP protocol.
Pup	PUP protocol.
Udp	UDP protocol.
Idp	IDP protocol.
IPv6	IPv6 protocol.
Raw	Raw IP protocol.
Ipx	IPX protocol.
Spx	SPX protocol.
SpxII	SPXII protocol.

A complete list of supported protocols you can find in MSDN.

Depending on the object meaning of Socket class, sockets are divided into active and passive.

- ***Active socket*** is designated for establishing a connection with a remote server from a client side.
- ***Passive socket*** is a server socket waiting for a connection from the clients.

Depending on the socket purpose (active or passive) the working algorithm will differ.

Moreover, there is a notion of synchronous and asynchronous socket.

The point is that when arranging a messages exchange using sockets we should foresee a mechanism allowing providing the services to all connected clients. Using asynchronous sockets for resolving a mentioned problem a mechanism providing platform multi-task execution is usually used.

Compared to synchronous asynchronous sockets have a built-in multi-event processing mechanism, which allows exercising a data exchange in an asynchronous mode.

Establishing the connection from the synchronous connection client side using TCP protocol

In order to establish a connection from the client side it is necessary to follow the steps:

1. To create a Socket type object with the assignation of its network type (in the below example we can ser Address-Family.InterNetwork (IPv4), transport protocol of Sock-etType.Stream type (TCP) and ProtocolType);
2. To request a Connect method, having passed him an IP-EndPoint class object in a capacity of a class object pa-

rameter, which encapsulates an IP address of a remote machine and a port that we need to connect to.

3. In case of successful connection we can initiate a message exchange in accordance with the protocols of the application, presentation and session layers using a Send method for sending messages and a Receive method for their accept.

```
IPAddress ip=IPAddress.Parse("207.46.197.32");
IPEndPoint ep = new IPEndPoint(ip, 80);
Socket s = new Socket(AddressFamily.InterNetwork,
           SocketType.Stream, ProtocolType.IP);
try
{
    s.Connect(ep);
    if (s.Connected)
    {
        String strSend="GET\r\n\r\n";
        s.Send(System.Text.Encoding.ASCII.
               GetBytes(strSend));
        byte[] buffer = new byte[1024];
        int l;
        do
        {
            l = s.Receive(buffer);
            textBox1.Text +=
                      System.Text.Encoding.ASCII.
                      GetString(buffer, 0, l);
         } while (l > 0);
     }
     else
         MessageBox.Show("Error");
     }
      catch (SocketException ex)

      MessageBox.Show(ex.Message);
}
```

In the above mentioned example we have made a connection to the port 80 (standard port of application protocol http) of a remote host 207.46.197.32 (Microsoft.com) and we have sent there a standard GET request (in accordance with the HTTP1.0 specification). The server has returned us the contents of the default requested page (protocol HTTP1.0 does not support the work with virtual hosts).

Pay your attention that we had to recode the strings into the byte arrays in ASCII encoding, because a data transmitted format is defined at the presentation layer of HTTP protocol.

After we have worked with the socket we have to close it properly, in order to do that, we need to sequentially request a Shutdown method for data transmission block and then Close — for releasing managed and unmanaged resources that socket uses.

Shutdown method accepts SocketShutdown listing in a capacity of a parameter, depending on the value that prohibits the further sending or (and) receiving messages.

To properly terminate working with the socket we add the following code block:

```
    finally{
                s.Shutdown(SocketShutdown.Both);
                s.Close();
            }
```

Accept of the connection from the server side in synchronous mode under an established connection (TCP): (working with a passive socket in synchronous mode)

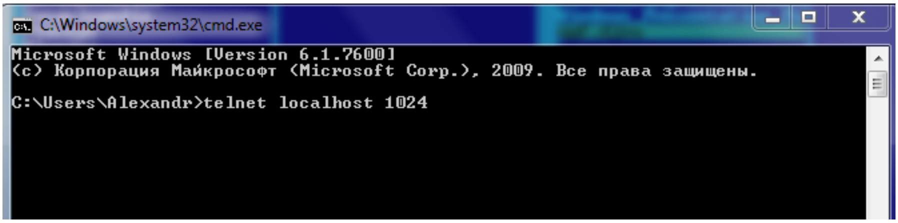
In order to create a socket accepting a TCP connection on the server side and provide data exchange with the connected client it is necessary to follow the steps below:

1. To create an object of Socket type with assignation of a network type (in the below example AddressFamily.InterNetwork (IPv4), types of transport protocol SocketType.Stream (TCP) and ProtocolType);
2. To bind a received socket with an IP address and port at the server having requested a socket Bind method. В качестве параметра Bind accepts a class IPEndPoint object in a capacity of the parameter, it also encapsulates an IP address of the server and the port clients have to connect to.
3. To put a socket into a listening state having requested a Listen method and having passed a size of a pending processing connections queue in a capacity of the parameter.
4. Inside of the cycle we request an Accept method, and its request blocks this execution flow. Accept method upon connecting the listening, client will unblock the flow and brings its new initiated Socket object, through which messages exchange with the remote client take place. Upon this the port number of an initiated socket differs from the port number through which it realizes.
5. After the termination of message exchange through a socket, which was received from an Accept method, it closes and then an Accept method is requested before the connection of the next client.

In the following example there is a pending connection to the port 1024 to an IP 127.0.0.1 (localhost), after the connection of a client he gets a string containing the current date and time, and client's IP address and port number of a remote client are displayed in a console window:

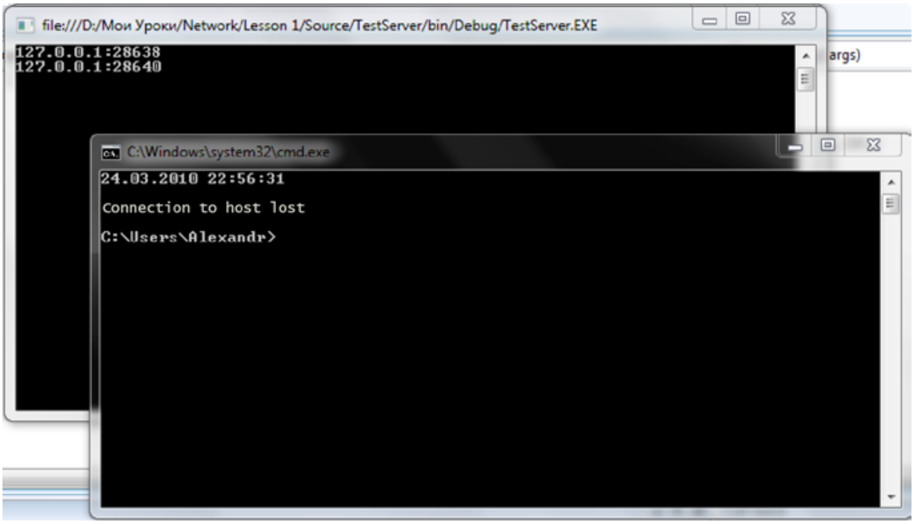
```
Socket s = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.IP);

IPAddress ip = IPAddress.Parse("127.0.0.1");
IPEndPoint ep = new IPEndPoint(ip, 1024);

s.Bind(ep);
s.Listen(10);

try
{
    while (true)
    {
       Socket ns = s.Accept();
       Console.WriteLine(ns.RemoteEndPoint.ToString());

       ns.Send(System.Text.Encoding.ASCII.
          GetBytes(DateTime.Now.ToString()));
       ns.Shutdown(SocketShutdown.Both);
       ns.Close();
    }
}
catch (SocketException ex)
{
    Console.WriteLine(ex.Message);
}
```

We can test the software functioning by means of telnet utility, having connected the localhost (127.0.0.1) to the port 1024.



```
Microsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\Alexandr>telnet localhost 1024
```

Here is an approximate look of the working server and a client


file:///D:/Мои Уроки/Network/Lesson 1/Source/TestServer/bin/Debug/TestServer.EXE
args)
C:\Windows\system32\cmd.exe

```
127.0.0.1:28638
127.0.0.1:28640
```

```
24.03.2010 22:56:31
Connection to host lost
C:\Users\Alexandr>
```

Naturally, if you indicate an IP address of your network interface in a Bind method, then the connection will be established also through this address.

7. Operation in Asynchronous Mode

This simple method of interaction between a client and a server, as it is shown in the previous example has the right for existence, though it has a list of disadvantages, for example if a client has to maintain connection with a server for quite a long time, then no one will be able to connect the server before the client releases the socket.

Though, there is an obvious solution of a problem, it is necessary to use multi-threading, i.e. interaction with each client has to be exercised in a separate execution flow of, that is, to use asynchronous processing. Of course, it is possible to implement manual multithread processing, as it is shown in the example below:

```
class Server
{
    delegate void ConnectDelegate (Socket s);
    delegate void StartNetwork(Socket s);
    Socket socket;
    IPEndPoint endP;

    public Server(string strAddr, int port)
    {
        endP = new IPEndPoint(IPAddress.
                Parse(strAddr), port);
    }

    void Server_Connect(Socket s)
    {
        s.Send(System.Text.Encoding.ASCII.
               GetBytes(DateTime.Now.ToString()));
        s.Shutdown(SocketShutdown.Both);
        s.Close();
    }

    void Server_Begin(Socket s)
    {
        while (true)
        {
            try
            {
                while (s!=null)
                {
                    Socket ns = s.Accept();
                    Console.WriteLine(ns.
                        RemoteEndPoint.ToString());
                    ConnectDelegate cd = new
                        ConnectDelegate(
                        Server_Connect);
                    cd.BeginInvoke(ns, null, null);
                }
            }
            catch (SocketException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }

    public void Start()
    {
        if (socket != null)
            return;
        socket = new Socket(AddressFamily.
                 InterNetwork, SocketType.Stream,
                 ProtocolType.IP);
        socket.Bind(endP);
        socket.Listen(10);
        StartNetwork start = new
                          StartNetwork(Server_Begin);
        start.BeginInvoke(socket, null, null);
    }

    public void Stop()
    {
        if (socket != null)
        {
            try
            {
                socket.Shutdown(SocketShutdown.Both);
                socket.Close();
                socket = null;
            }
            catch (SocketException ex)
            {
            }
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Server s = new Server("127.0.0.1", 1024);
        s.Start();
        Console.Read();
        s.Stop();
    }
}
```

The example demonstrates that during establishing a connection from a client side there forms a delegate for handling the connection with a client. And this delegate moves in an asynchronous mode, which provides execution of the interaction with a connected client in a separate flow. And the listening socket is also launched in a separate flow.

Did it seem to you that the above mentioned code was too difficult? Hope not. But though, it can be simplified. The thing is that sockets can independently work in asynchronous mode without writing of an additional code.

To provide the possibility of asynchronous work in the class Socket there is a set of relevant methods:

Name	Description
AcceptAsync	Initiates an asynchronous operation in order to attempt an input connection.
BeginAccept	Overloaded. Initiates an asynchronous operation in order to attempt an input connection.
BeginConnect	Overloaded. Initiates an execution of a asynchronous request in order to connect a remote node
BeginDisconnect	Initiates an execution of an asynchronous request in order to disconnect from a remote terminal.
BeginReceive	Overloaded. Initiates an execution of an asynchronous data accept from a connected Socket object.
BeginReceiveFrom	Initiates an execution of an asynchronous data accept from an indicated networking device.
BeginReceiveMessageFrom	Initiates an asynchronous receipt of a preset number of the data bytes into a designated place of a data buffer using a predefined object SocketFlags, as well as saves a terminal and package information.
BeginSend	Overloaded. Executes an asynchronous data transmission to a connected Socket object.

Name	Description
BeginSendFile	Overloaded. Executes an asynchronous file transmission to a connected Socket object.
BeginSendTo	Executes an asynchronous data transmission to a designated node.
ConnectAsync	Initiates an execution of an asynchronous request in order to connect a remote node.
DisconnectAsync	Initiates an execution of an asynchronous request in order to disabling from a terminal.
EndAccept	Overloaded. Asynchronously receives an attempt of input connection.
EndConnect	Terminates a pending asynchronous request for the connection.
EndDisconnect	Terminates a pending disconnection asynchronous request.
EndReceive	Overloaded. Terminates a delayed asynchronous reading.
EndReceiveFrom	Terminates a delayed asynchronous reading from a particular terminal.
EndReceiveMessageFrom	Terminates a delayed asynchronous reading from a particular terminal. This method also shows more information about a package, then EndReceiveFrom method.
EndSend	Overloaded. Terminates a delayed operation on asynchronous transmission.
EndSendFile	Terminates a delayed operation on asynchronous file transmission.
EndSendTo	Terminates a delayed operation on asynchronous transmission to a designated destination.
ReceiveAsync	Initiates an execution of an asynchronous request in order to receive data from a connected Socket object.

Name	Description
ReceiveFrom	Overloaded. Receives a datagram and saves a source terminal point.
ReceiveFromAsync	Initiates an execution of an asynchronous data accept from an indicated networking device.
ReceiveMessageFrom	Accepts an indicated number of data bytes into a designate place of data buffer using a preset object SocketFlags, as well as saves a terminal and package information.
ReceiveMessageFromAsync	Initiates an asynchronous accept of a preset number of data bytes into a designate place of data buffer using a preset object SocketAsyncEventArgs. SocketFlags, as well as saves a terminal and package information.
SendAsync	Executes an asynchronous data transmission to a connected object Socket.
SendPacketsAsync	Executes an asynchronous transmission of files set or data buffers within the memory to a connected object Socket.
SendToAsync	Executes an asynchronous data transmission to a designated node.

Accept of a connection from a server side in an asynchronous mode with a connecting establishment (TCP): (working with a passive socket in an asynchronous mode)

In order to create a socket accepting a TCP connection on a server side and provide a data exchange with a connected client in asynchronous mode we have to follow the steps:

1. To create an object of Socket type with the indication of network type (in the below example AddressFamily. InterNetwork (IPv4), transport protocol type SocketType. Stream (TCP) and ProtocolType.IP);

2. To bind an accepted socket with an IP address and port on a server by means of socket Bind method. Bind accepts an object of IPEndPoint class in a capacity of the parameter, and encapsulates and IP address of the server and port clients are going to be connected to.
3. To put a socket into listening state by means of requesting a Listen method, passing it for connections handling.

First three paragraphs completely correspond to an operating procedure in a synchronous mode, and now we shift to the distinctions.

1. It is necessary to create a delegate of AsyncCallback type and request a BeginAccept method, passing a delegate and our listening socket in a capacity of the parameter.
2. When a client connects a delegate will be requested, the one to which our listening socket will come in a AsyncState property of the parameter of IAsyncResult type.
3. Within an accepted socket an EndAccept is requested, which returns a new Socket object, through which message exchange with a remote client takes place.
4. BeginAccept is requested again and the operation cycle is repeated.

Analogically we can also organize an asynchronous message sending to a client using BeginSend method, having transferred to him a delegate, which will be requested by the termination of the sending operation and a socket for data exchange with a client.

A below example launches connection wiretapping in an asynchronous mode and asynchronous message sending to a client:

```
class AsyncServer
{
    IPEndPoint endP;
    Socket socket;

    public AsyncServer(string strAddr, int port)
    {
        endP = new IPEndPoint(IPAddress.
               Parse(strAddr), port);
    }
    void MyAcceptCallbakFunction(IAsyncResult ia)
    {
        //get a link to the listening socket
        Socket socket=(Socket)ia.AsyncState;
        //get a socket to exchange data with
        //the client
        Socket ns = socket.EndAccept(ia);
        //output the connection information to the console
        Console.WriteLine(ns.RemoteEndPoint.ToString());

        //send the client the current time asynchronously,
        //by the end of the sending operation,
        //MySendCallbackFunction method will be called.
        byte[] sendBufer = System.Text.Encoding.
             ASCII.GetBytes(DateTime.Now.ToString());
        ns.BeginSend(sendBufer, 0, sendBufer.Length,
             SocketFlags.None, new
             AsyncCallback(MySendCallbackFunction), ns);

        //resume asynchronous Accept
        socket.BeginAccept(new
             AsyncCallback(MyAcceptCallbakFunction),
             socket);
    }

    void MySendCallbackFunction(IAsyncResult ia)
    {
        //after sending data to the client,
        //close the socket (if we would need to continue
        //data exchange, we could have arranged it here)
        Socket ns=(Socket)ia.AsyncState;
        int n = ((Socket)ia.AsyncState).EndSend(ia);
        ns.Shutdown(SocketShutdown.Send);
        ns.Close();
    }

    public void StartServer()
    {
        if (socket != null)
            return;
        socket = new Socket(AddressFamily.
                 InterNetwork, SocketType.Stream,
                 ProtocolType.IP);
        socket.Bind(endP);
        socket.Listen(10);
        //start asynchronous Accept,
        //when the client is connected,
        //the MyAcceptCallbakFunction
        //handler is called
        socket.BeginAccept(new
               AsyncCallback(MyAcceptCallbakFunction),
               socket);
    }
}

class Program
{
    static void Main(string[] args)
    {
        AsyncServer server = new
                AsyncServer("127.0.0.1", 1024);
        server.StartServer();
        Console.Read();
    }
}
```

Asynchronous programming of TCP socket at a client side can be exercised by means of BeginConnect method in order to create a connection from a client side in an asynchronous mode. Moreover, we can use an asynchronous messages transmission and accept on a client side in the same way as upon the programming on a server side.

Use of class Soket for working via UDP protocol in synchronous mode

If we use a UDP protocol, i.e. a protocol without establishing a connection, we can use SendTo methods for messages sending, and in order to receive datagrams we can apply ReceiveFrom methods.

Working algorithm with UDP listening to a socket is the following:

1. To create an object type Socket, having assigned his network type (in the below example AddressFamily.InterNetwork (IPv4), transport protocol type SocketType.Dgram (UDP) and ProtocolType.IP) ;

```
socket=new Socket(AddressFamily.InterNetwork,
           SocketType.Dgram,
           ProtocolType.IP);
```

2. To bind an received socket with an IP address and port on a server by means of requesting of socket Bind method. Bind accepts and object of IPEndPoint class in a capacity of the parameter and encapsulates and IP address of a server and port clients are going to be connected to.

```
socket.Bind(new IPEndPoint(IPAddress.
       Parse("10.2.21.129"), 100));
```

3. To request from a socket a ReceiveFrom method, and upon that an execution of the current flow will be suspended till the data emerges within an incoming buffer. ReceiveFrom returns a number of read bytes. If a size of a buffer for reading will be insufficient, then there can arise an exception SocketException;

```
int l=rs.ReceiveFrom(buffer, ref ep);
String strClientIP=((IPEndPoint)ep).Address.ToString();
    String str = String.Format("\nReceived from
                {0}\r\n{1}\r\n", strClientIP,
                System.Text.Encoding.Unicode.
                GetString(buffer, 0, l));
```

4. For data transmission we use a SendTo method (I will give a description of the method later, when we will review a client UDP socket);

So far as a call request ReceiveFrom is a blocking one, then we can bring out its handling into a separate execution flow, as it is shows in the example below.

```
delegate void AddTextDelegate(String text);
System.Threading.Thread thread;
Socket socket;

public Form1()
{
    InitializeComponent();
}
void AddText(String text)
{
    textBox1.Text+=text;
}
```

```
void RecivFunction(object obj)
{
    Socket rs=(Socket) obj;

    byte[] buffer = new byte[1024];
    do
    {
        EndPoint ep = new IPEndPoint(0x7F000000, 100);
        int l=rs.ReceiveFrom(buffer, ref ep);
        String strClientIP= ((IPEndPoint)ep).Address.
            ToString();
        String str = String.Format("\nReceived from
            {0}\r\n{1}\r\n",
        strClientIP, System.Text.Encoding.Unicode.
            GetString(buffer, 0, l));
        textBox1.BeginInvoke(new
            AddTextDelegate(AddText), str);
    } while (true);
}

private void button1_Click(object sender, EventArgs e)
{
    if (socket != null &&thread!=null)
        return;
    socket=new Socket(AddressFamily.InterNetwork,
            SocketType.Dgram, ProtocolType.IP);
    socket.Bind(new IPEndPoint(IPAddress.
            Parse("10.2.21.129"), 100));

    thread = new System.Threading.Thread(RecivFunction);
    thread.Start(socket);
}

private void button2_Click(object sender, EventArgs e)
{
    if (socket != null)
    {
        thread.Abort();
        thread = null;
        socket.Shutdown(SocketShutdown.Receive);
        socket.Close();
        socket = null;
        textBox1.Text = "";
    }
}

Messages sending via UDP protocol doesn't cause
        any questions:
private void button4_Click(object sender, EventArgs e)
{
    Socket socket = new Socket(AddressFamily.
        InterNetwork, SocketType.Dgram,
        ProtocolType.IP);

    socket.SendTo(System.Text.Encoding.Unicode.
        GetBytes(textBox2.Text),
        new IPEndPoint(IPAddress.
        Parse("10.2.21.255"), 100));

    socket.Shutdown(SocketShutdown.Send);
    socket.Close();
}
```

Here we only have to create a socket and send a synchronous message using a SendTo to a designated IP address.

Pay attention that I have indicated a broadcasting address of my sub-network in a capacity of a target node, meaning that my messages are addressed to all sub-network nodes.

Working with UDP in an asynchronous mode

Beside a synchronous messages sending, a UDP socket also supports a possibility of operating in an asynchronous mode.

For an asynchronous reading from the UDP socket we use a BeginReceiveFrom method, and for a synchronous sending — a BeginSendTo.

Below I will give an example of asynchronous messages accept via UDP socket:

```
private void button1_Click(object sender, EventArgs e)
{
    if (socket != null)
        return;
    socket=new Socket(AddressFamily.InterNetwork,
          SocketType.Dgram, ProtocolType.IP);
    socket.Bind(new IPEndPoint(IPAddress.
          Parse("10.2.21.129"), 100));
    state.workSocket = socket;
    RcptRes = socket.BeginReceiveFrom(state.buffer,
        0,
        StateObject.BufferSize,
        SocketFlags.None,
        ref ClientEP,
        new AsyncCallback(Receive_Completed), state);
}

void Receive_Completed(IAsyncResult ia)
{
    try
    {
        StateObject so = (StateObject)ia.AsyncState;
        Socket client = so.workSocket;
        if (socket == null)
            return;
        int readed = client.EndReceiveFrom(RcptRes,
                ref ClientEP);

        String strClientIP = ((IPEndPoint)ClientEP).
                Address.ToString();
        String str = String.Format("\nReceived from
                {0}\r\n{1}\r\n",
            strClientIP, System.Text.Encoding.
                Unicode.GetString(so.buffer, 0,
                readed));
        textBox1.BeginInvoke(new
                AddTextDelegate(AddText), str);
        RcptRes = socket.BeginReceiveFrom(state.buffer,
            0,
            StateObject.BufferSize,
            SocketFlags.None,
            ref ClientEP,
            new AsyncCallback(Receive_Completed),
            state);
    }
    catch (SocketException ex)
    {
    }
}
```

Sending asynchronous messages via UDP socket
    is not difficult:

```
private void button4_Click(object sender, EventArgs e)
{
    Socket socket = new Socket(AddressFamily.
            InterNetwork,
            SocketType.Dgram,
            ProtocolType.IP);

    byte[] buffer=System.Text.Encoding.Unicode.
            GetBytes(textBox2.Text);
    SendRes = socket.BeginSendTo(buffer, 0,
            buffer.Count(), SocketFlags.None,
            (EndPoint)new IPEndPoint(IPAddress.
            Parse("10.2.21.255"), 100),
            new AsyncCallback(Send_Completed), socket);
}
```

```
void Send_Completed(IAsyncResult ia)
{
    Socket socket = (Socket)ia.AsyncState;
    socket.EndSend(SendRes);
    socket.Shutdown(SocketShutdown.Send);
    socket.Close();
}
```

And in the end I would mention that we have got used to address node names not using an IP address, but a DNS node name. For a node name resolution into an IP address (and vice versa) we could use a static class System.Net.Dns, and you will get acquainted with it closer in MSDN.

In this respect, method of the class GetHostAddresses returns an array of IP addresses, linked with the domain name.

8. Home Assignment

1. To write network clock using datagram protocol. Here is a service of package sending to LAN installed on a server, and the packages contain information about the current time. Clients display the current time. You can also make a call system optionally.
2. Write a messages exchange system between the clients. The system should use a central server to which clients connect and leave there their messages. A client that to whom messages are addressed которому will receive them from a server upon a request. The system should apply a protocol oriented to the connection (TCP).