***Week 05:*Introduction to Network Programming**

- What is network programming?
- Advantages of network programming
- Overview of name spaces
- What are sockets?
  - Connection oriented sockets
  - Connection less sockets
- System. NET
- System. Net. Sockets
- Over view of TCP and UDP
  - TCP Programming features
  - UDP Programming features

Lecture

What is Network Programming?

It is a field of programming which enables two or more hosts to communicate with each other.Network programming is the writing applications (programs) for sending and receiving data across network.

The Internet Protocol (IP) is at the core of network programming. IP is the vehicle that transports data between systems, whether within a local area network (LAN) environment or a wide area network (WAN) environment. Though there are other network protocols available to the Windows network programmer, IP provides the most robust technique for sending data between network devices, especially if they are located across the Internet.

Programming using IP is often a complicated process. There are many factors to consider concerning how data is sent on the network:

- the number of client and server devices,
- the type of network,
- network congestion, and
- error conditions on the network

Because all these elements affect the transportation of your data from one device to another, understanding their impact is crucial to your success in network programming.

The Windows OS family offers many ways to determine IP configuration information, both manually and from within a program. [*must read Chapter 2:* ***Finding IP Address Information***]

The **ipconfig** program displays IP network information for each active network interface on the system. The default form of **ipconfig** displays basic information for each network device:

```
C:\>ipconfig
```

To obtain more detailed information about a device, use the /all switch for the ipconfig command.

```
C:\>ipconfig /all
```

Assignment: What are Advantages of network programming?

Overview of Namespaces

Namespaces are used to organize the classes. It helps to control the scope of methods and classes in larger .Net programming projects.It is also referred as named group of classes having common features. The members of a namespace can be *namespaces*, *interfaces*, *structures*, and *delegates*.

Namespaces have the following properties:

- They organize large code projects.
- They are delimited by using the . (dot) operator.
- The **using** directive obviates the requirement to specify the name of the namespace for every class.
- The **global** namespace is the "root" namespace: **global::System**will always refer to the .NET System namespace.

Namespaces are heavily used in C# programming in two ways.

1. First, the .NET Framework uses namespaces to organize its many classes, such as:

```
System.Console.WriteLine("C# network programming");
```

System is a namespace and *Console* is a class in that namespace. The using keyword can be used so that the complete name is not required, for example:

```
using System;
. . .
```

```
Console.WriteLine("C# network programming");
```

2. Second, declaring your own namespaces can help you control the scope of class and method names in larger programming projects. Use the **namespace** keyword to declare a namespace.

Syntax:

```
namespace name_of_namespace{

// Namespace (Nested Namespaces)
// Classes
// Interfaces
// Structures
// Delegates
}
```

What are Sockets?

[Blum, Chapter 3:]

A network file descriptor is called a *socket*. UNIX and Windows network programs both utilize sockets for all network communication. (Starting in the 4.2BSD UNIX release, network access was also defined using file descriptors.) In Windows environment,**Winsock** is the implementation of sockets. Winsock is the base on which the C# Socketclass implementation is built.

In socket-based network programming, you do not directly access the network interface device to send and receive packets. Instead, an intermediary file descriptor is created to handle the programming interface to the network.

(A file descriptor is an abstract indicator used to access a file or other input/output resource, such as a pipe or network socket.A file descriptor is a number that uniquely identifies an open file in a computer's operating system. It describes a data resource, and how that resource may be accessed.)

The special file descriptors used to reference network connections are called *sockets*. The socket defines the following:

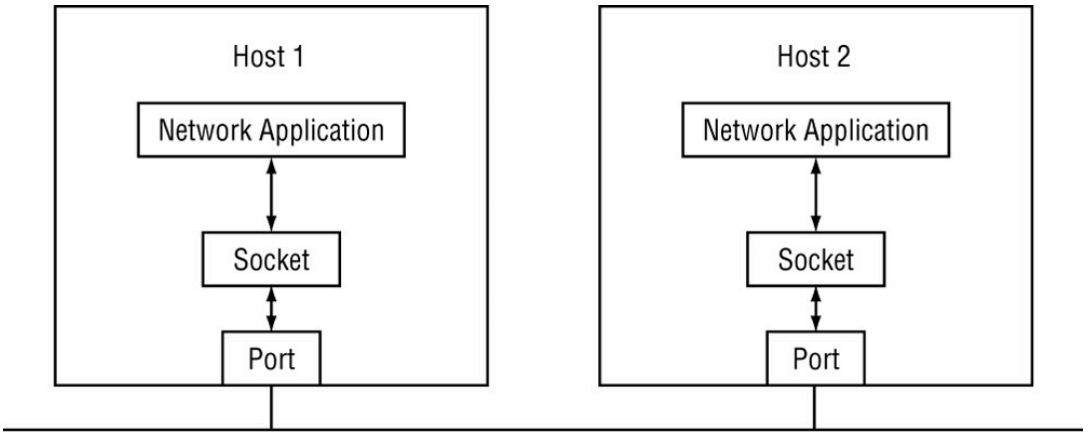
- A specific communication domain, such as a network connection or a Unix Interprocess Communication (IPC) pipe
- A specific communication type, such as stream or datagram
- A specific protocol, such as TCP or UDP

After the socket is created, it must be bound to either

- a specific network address and port on the system, or

- to a remote network address and port.

Once the socket is bound, it can be used to send and receive data from the network.

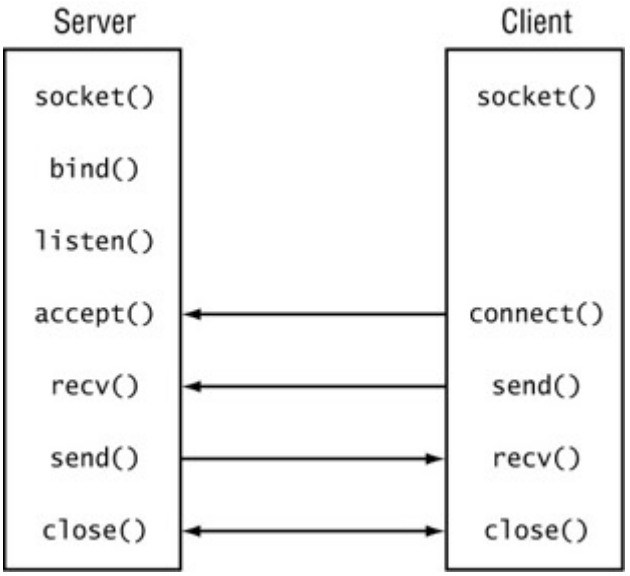


The world of IP connectivity revolves around two types of communication:

- connection oriented and
- connectionless

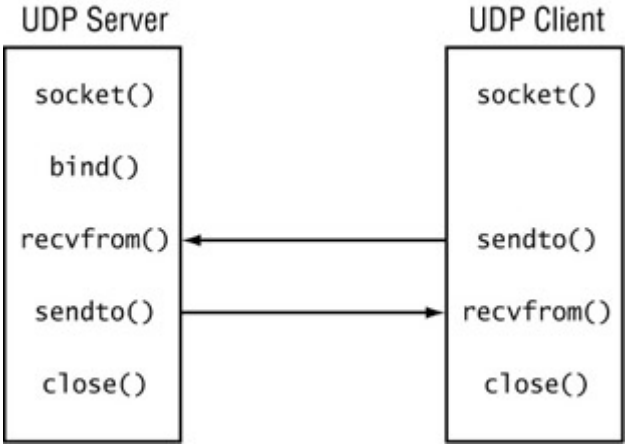
Connection-Oriented Sockets

In a connection-oriented socket (one that uses the SOCK_STREAM type) the TCP protocol is used to establish a session (connection) between two IP address endpoints. There is a fair amount of overhead involved with establishing the connection, but once it is established, data can be reliably transferred between the devices. To create a connection-oriented socket, separate sequences of functions must be used for server programs and for client programs.



Connectionless Sockets

SOCK_DGRAM type sockets use the UDP protocol; therefore, no connection information is required to be sent between network devices. Because of this, it is often difficult to determine which device is acting as a "server", and which is acting as a "client". If a device is initially waiting for data from a remote device, the socket must be bound to a local address/port pair using the **bind()** function. Once this is done the device can send data out from the socket, or receive incoming data from the socket. Because the client device does not create a connection to a specific server address, the **connect()** function need not be used for the UDP client program.



Socket Programming in Windows **** optional

Windows Socket Functions

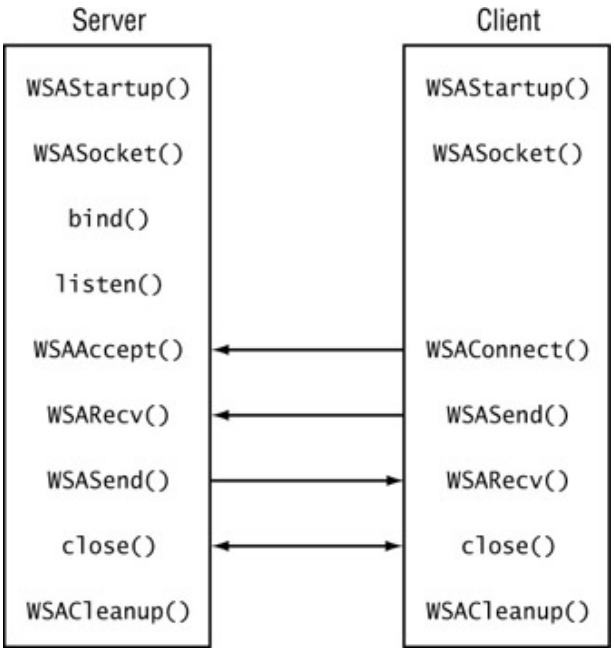
Windows network programming model is derived from the comparable Unix model. Many features of the Windows operating systems have their roots in Unix systems. Much of Windows network programming was modeled after the Unix Berkeley socket method. It was called, not surprisingly, Windows Sockets, or Winsock for short. The Winsock interface was designed to allow network

programmers from the Unix environment to easily port existing network programs, or to create new network programs in the Windows environment without a large learning curve.

The Winsock APIs were implemented as a set of header and library files for developers and DLL files to be used by applications.

The core of the Winsock environment is, of course, the socket. Just as in Unix, all Windows network programs create a socket to establish a link with the underlying network interface on the Windows system. All of the standard socket function calls employed in the Unix world were ported to the Windows system. However, there are a few differences between Unix sockets and Winsock.

The Winsock WSA programming functions for servers and clients



Overview of TCP and UDP

[Blum: Chapter2]

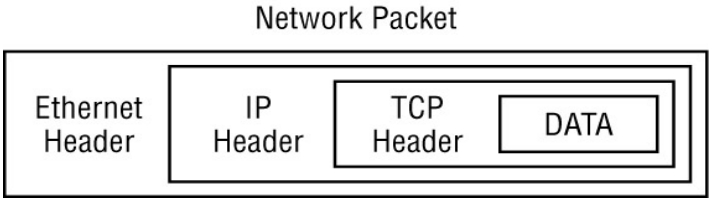


Figure 1: Network protocol layers in packets

The network packets contain several layers of information to help the data get transported between the two network devices. Each layer of information contains bytes arranged in a predetermined order, specifying parameters specific for the protocol layer. Most packets in IP network programming will contain three separate protocol layers of information, along with the actual data being transmitted between network devices.

Ethernet Header: The Ethernet header shows the Media Access Card (MAC) addresses that are used to identify individual devices on the Ethernet network, along with an Ethernet protocol number that identifies the next layer protocol contained in the Ethernet packet. Each Ethernet packet conforms to a predetermined layout:

- A 6-byte destination Ethernet (or MAC) address
- A 6-byte source Ethernet (or MAC) address
- A 2-byte next-layer protocol identifier
- A data payload of 46 to 1500 bytes
- A 4-byte checksum

IP Header: some important fields in IP layer include version, header length, type of service, address fields, protocoletc.

TCP Header:fields contain the information necessary for TCP to implement its advanced connection and data reliability features.Each of the many fields in the TCP Header is associated with a particular function in the TCP session.

Data:contains actual data

Overview of TCP

The Transmission Control Protocol (TCP) adds connection information to the data packet. This allows programs to create an end-to-end connection between two network devices, providing a consistent path for data to travel. TCP guarantees the data will reliably be delivered to the destination device or that the sender will receive indication that a network error occurred.

Because of this feature, TCP is called a connection-oriented protocol. Each TCP connection, or session, includes a certain amount of packet overhead related to establishing the connection

between the two devices. Once the connection is established, data can be sent between the devices without the application having to check for lost or out-of-place data.

TCP uses ports to identify individual TCP connections on a network device. The port value identifies a TCP endpoint on the device to be used for a particular application. To communicate with an application on a remote device, you must know two pieces of information:

- The remote device's IP address
- The TCP port assigned to the remote application

For a TCP connection to be established, the remote device must accept incoming packets on the assigned port.*The combination of an IP address and a port number defines an****IP endpoint***.The IANA has defined a list of standard TCP ports assigned to specific applications.Ports that are numbered from ***0*** to ***1023*** are called *well-known* ports because they are assigned to specific, common applications.Ports numbered from ***1024*** to ***65,535*** are open for use by any application.

Q. List some well-known TCP application port numbers.

Overview of UDP

The User Datagram Protocol (UDP) is another popular high-level protocol used in IP communications. Unlike TCP, UDP provides a *connectionless* path between network devices to transmit data, and thus does not need all of the overhead of session establishment flags and connection states. Each UDP "session" is nothing more than a single packet transmitted in one direction. The UDP header fields are pretty straightforward:

- Source Port
- Destination Port
- Message Length
- Checksum
- Next Level Protocol

UDP tracks individual connections using port numbers and assigns port numbers from 0 to 1023 to reserved application ports. Ports 1024 to 65536 are available for you to use in your applications.

Table 2.12: Well-known UDP Port Numbers

Port	Description	Port	Description
53	Domain Name System	137	NetBIOS name service
69	Trivial File Transfer Protocol	138	NetBIOS datagram
111	Remote Procedure Call	161	Simple Network Management Protocol

TCP Programming features

Read [Blum: chapter 2]

UDP Programming features

Read [Blum: chapter 2]

References

- Network programming by Richard Blum
- https://www.geeksforgeeks.org/c-sharp-namespaces/
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/namespaces/