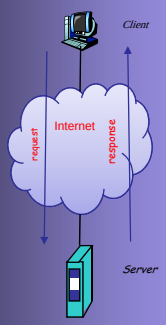


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 5.1: C# TCP C/S Programming (Part 1)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

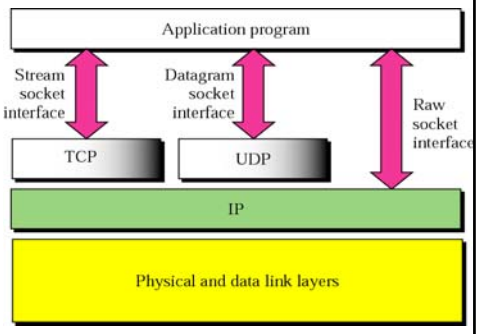
- Understand the basic underlying concepts of C/S programming – sockets, ports, TCP, UDP
- Learn C# basic classes for writing C/S applications
- Learn how to write a TCP server using TcpListener class
- Learn how to test a TCP server using Telnet
- Learn how to write a TCP client using TcpClient class

Basic Concepts

- Involves writing two programs – a **Server** and a **Client**
- Communication between them is achieved using the socket programming interface (provided by all modern Operating Systems).
  - It provides methods that a local process calls to communicate with a remote process
- Two approaches in .NET to access the socket interface
  - Using the Socket class to program both Servers and Clients.
  - Using TcpListener, TcpClient and UdpClient
    - Much easier than the Socket class so we start writing programs with this high level classes.

Basic Concepts (cont.)

- Before writing a C/S application, decide which transport layer protocol to be used (TCP, UDP, or none)
- Types of Sockets
  - Stream socket
    - To be used with a connection-oriented protocol such as TCP
  - Datagram socket
    - To be used with a connectionless protocol such as UDP
  - Raw socket
    - Directly use services of IP

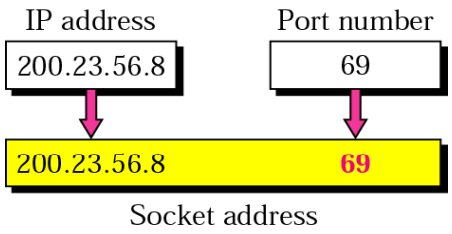


TCP vs. UDP

- TCP
  - Stands for Transmission Control Protocol
  - Connection oriented (i.e. a connection is first established before data exchange begins)
  - Data bytes are delivered as streams (in sequence)
  - Provides Error and Flow control to ensure data reaches its target reliably.
  - Connection is terminated once one of the communicating devices requests that
  - Drawback: Slow especially if the network is not perfect
- UDP
  - Stands for User Datagram Protocol
  - Connectionless (i.e. no connection is established at all)
  - Data is delivered is packets. Packet are routed over the network until they reach their target.
  - No sequencing or any form of error control is provided
  - Thus, packets may reach their target out of sequence or may not reach at all, i.e. Unreliable.
  - Advantage : Fast; thus more suitable for real-time applications

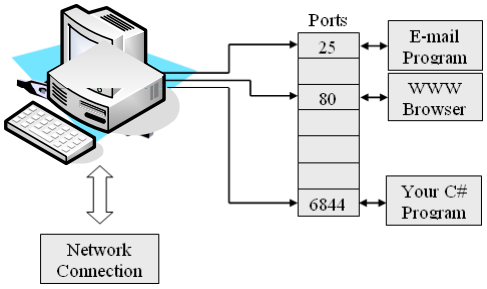
Basic Concepts (cont.)

- When programming a Server or a Client, we must associate it with a Socket Address (also known as EndPoint)
- EndPoint is a combination of IP address and Port number



Basic Concepts (cont.)

- The IP addresses identify communicating nodes, while port numbers identify communicating applications.
- A port number is a 16 bit number; ranges from 0 to 65,535
  - Used to uniquely identify a specific application running on a node
- Port numbers 0 to 1023 are reserved for well-known services (hence do not use them for your programs).



Basic Concepts (cont.)

- Some of the most common TCP port numbers are as follows:

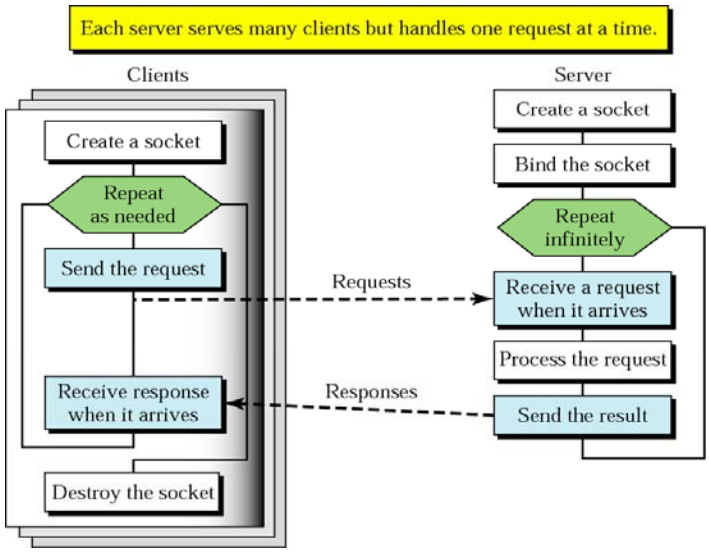
Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Basic Concepts (cont.)

- Some of the most common UDP port numbers are as follows:

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Programming C/S Application



Basic Network Programming Classes

- .NET provides many network programming classes in System.Net and System.Net.Sockets namespaces.
- Main classes needed to write simple TCP Server and Client:
  - IPAddress,
  - IPEndPoint
  - TcpListener
  - TcpClient

IPAddress Class

- Used to represent IP address as an object.
- Constructors:

  public IPAddress(long address)

  public IPAddress(byte[] address)
- These constructors are rarely used, since we hardly have the IP address represented in bytes or long format.
- Has the following properties:
  - **Any**
    - Returns the IP address 0.0.0.0 . Normally used for a server. Sever must listen for clients on all its network interfaces.
  - **Broadcast**
    - Returns the broadcast address: 255.255.255.255
  - **Loopback**
    - Returns the loopback address : 127.0.0.1

IPAddress Class (cont.)

- **Parse():** a static method that takes an IP address as a string and returns an IPAddress object
  - is often used to create an instance of **IPAddress**
- Signature

  static IPAddress Parse(String address)
- Example

```
IPAddress ip = IPAddress.Parse("127.0.0.1");
```

NOTE : Parse method does not accept a domain name. To get an IP address from a domain name, you must use a method of the Dns class.

Basic Network Programming Classes…

- Another static method, ***IsLoopback()*** can be used to test if an address is a loop back address

static bool IsLoopback(IPAddress address)	Returns true if *address* is a loop back address

The IPEndPoint Class

- To represent IPAddress and Port number as a single object.
- The most useful constructor of this class has the form:

```
public IPEndPoint(IPAddress address, int port)
```

e.g: IPEndPoint endPoint = IPEndPoint(IPAddress.Any, 9999);

- The class has properties **IPAddress** and **Port**, that can be used to get the corresponding components of the end-point.
- Note: IPEndPoint is a derived class of the abstract class, **EndPoint**. The other derived class is **IrdAEndPoint**.

Basic Network Programming Classes…

The TcpListener Class:

- This is used to write a basic TCP Server program.

Method	Description
TcpListener (IPAddress, int port)	Creates and Binds the server to a specific IPAddress object and port number
TcpListener (IPEndPoint ie)	Creates and Binds the server to an IPEndPoint object
void Start()	Starts the server – put it in listen mode
TcpClient AcceptTcpClient()	Accepts connection from a TcpClient
void Stop()	Stops the server
bool Pending()	Determines if there are pending connections

Basic Network Programming Classes…

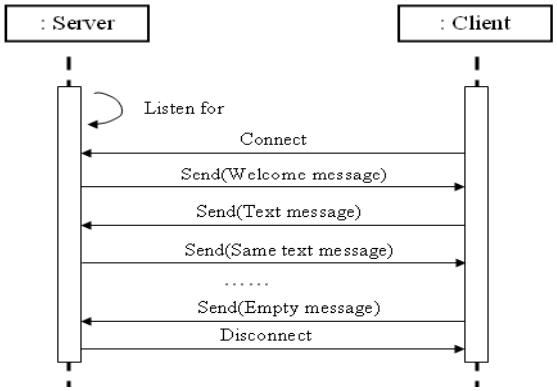
TcpClient Class:

- This is used to Write a basic TCP Client program.

Method	Description
TcpClient()	Creates an instance of TcpClient
TcpClient(IPEndPoint localEP)	Creates instance of TcpClient and binds it to a Local end point
TcpClient(string hostname, int port)	Creates instance of TcpClient and connects it to a Remote end point
void Connect(IPEndPoint) **void Connect(IPAddress, int)** **void Connect(string, int)**	Connects the client to a server. This is only necessary if the client is created using one of the first two constructors.
NetworkStream GetStream()	Returns a NetworkStream from the client's underlying socket.
void Close()	Closes the connection and releases all resources

Example

- Echo Client/Server Application
  - An important thing that must be decided when writing a network program is the **protocol** - the rule that governs communication between the server and the client.
  - In this example, the server will communicate with a client under a protocol summarized using the sequence diagram:



Programming a Server Application

- The following algorithm shows the process involved in writing a typical TCP server application.
  - a) Create the Server object - e.g using TcpListener class.
  - b) Bind the Server to a specific local IPEndPoint.
  - c) Place the Server in passive (listening) mode.
  - d) Accept the next connection request from a client.
  - e) Send acceptance indication to client – welcome
  - f) Repeat: Read a request, process the request, and send back the results.
  - g) Close the connection when done with a client.
  - h) Return to d) for next client.

Programming a Server Application …

- The following is an Echo Server using TcpListener class.

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.IO;

5.  class SimpleTcpServer {
6.    public static void Main() {
7.      TcpListener server = new TcpListener(IPAddress.Any, 9050);
8.      server.Start();
9.
10.     Console.WriteLine("Waiting for Client...");
11.     TcpClient client = server.AcceptTcpClient();
12.     Console.WriteLine("Connected with a client");
13.
14.     NetworkStream stream = client.GetStream();
15.     StreamReader reader = new StreamReader(stream);
16.     StreamWriter writer = new StreamWriter(stream);
```


Programming a Server Application …

```
17.     writer.WriteLine("Welcome to my test server");
18.     writer.Flush();
19.     String line = null;
20.     while((line = reader.ReadLine()).Length != 0) {
21.       Console.WriteLine(line);
22.       writer.WriteLine(line);
23.       writer.Flush();
24.     }
25.     client.Close();
26.     server.Stop();
27.   }
28. }
```

Using Telnet to test a TCP server

- Most modern operating systems provide a simple general TCP client application, Telnet, that can be used to test TCP servers.
- Telnet comes with all Windows OS platforms.
- To start Telnet, simply go command window and type:

  C:\>telnet *ipaddress port*

  where **ipaddress** and **port** are the IP address and the port number on which the server is listening

Programming a Client Application

- The following algorithm shows the process involved in writing a TCP client application.
  - a) Create a Client object – e.g using TcpClient class.
  - b) Send a Connect-request to a server listening at a specific EndPoint.
  - c) Receive response - usually a welcome message.
  - d) Repeat : Send a service request, receive and process the response.
  - e) When done, notify server of intention to disconnect.
  - f) Close the connection.

Programming a Client Application …

- The following is an Echo Client using TcpClient class.

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.IO;

5.  class SimpleTcpClient {
6.    public static void Main() {
7.      TcpClient client = new TcpClient("localhost", 9050);

8.      NetworkStream stream = client.GetStream();
9.      StreamReader reader = new StreamReader(stream);
10.     StreamWriter writer = new StreamWriter(stream);
11.
12.     String input = reader.ReadLine();
13.     Console.WriteLine(input);
```


Programming a Client Application …

```
14.     String line = null;
15.     do {
16.       Console.Write("Enter Message for Server Enter to Stop: ");
17.       line = Console.ReadLine();
18.       writer.WriteLine(line);
19.       writer.Flush();
20.       if (line.Length != 0) {
21.            line = "Echo: "+ reader.ReadLine();
22.            Console.WriteLine(line);
23.       }
24.     } while(line.Length != 0);
25.     client.Close();
26.   }
27. }
```

Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
  - Data Communications and Networking, 4/e. Behrouz A Forouzan, McGraw-Hill Higher Education
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm