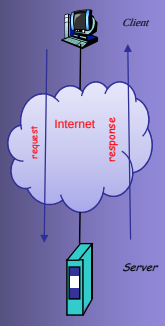


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 6.1: C# UDP 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 UdpClient class
- Learn how to write UDP server and client using the UdpClient class
- Learn how to write UDP server and client using the Socket class

The UdpClient class

- **UdpClient** class is designed to simplify the process of creating UDP applications.
- Since UDP is connectionless, there is no listener component.
  - Both server and client applications are created using the same UdpClient class.

UdpClient()	Bound the new instance to any local IP and any local port
UdpClient(int port)	Bounds the new instance to a specific local port and any local IP
UdpClient(IPEndPoint localEP)	Bounds the new instance to a specific local IP endpoint
UdpClient(string host, int port)	Bounds the new instance to any local endpoint and associate it with a default remote endpoint

The UdpClient class …

- The following are the methods of the **UdpClient** class

void Connect(IPEndPoint remoteEP) void Connect(IPAddress ip, int port) void Connect(String host, int port)	Establishes a default remote host and default remote port
public byte[] Receive(ref IPEndPoint remoteEP)	Returns a UDP datagram sent by a remote host.
int Send(byte[] data, int size) int Send(byte[] data, int size, IPEndPoint remoteEP) int Send(byte[] data, int size, string host, int port)	Sends datagrams to the specified remote node. The first version assumes a default remote end-point has been established.
void Close()	Closes the UDP connection

The UdpClient class …

Notes about some of the methods of UdpClient

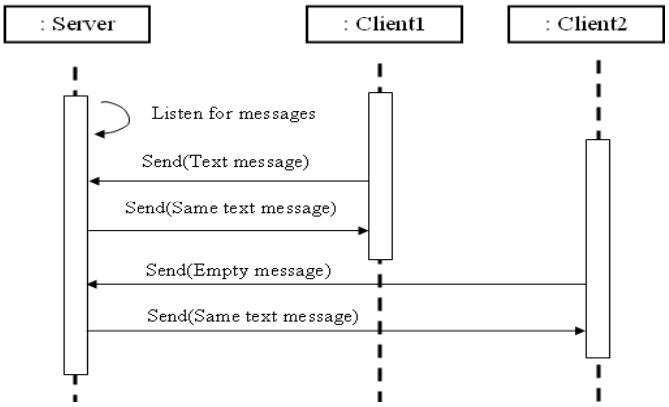
- Why Connect method since UDP is connection-less?
  - It establishes a default remote end-point, so that the end-point does not need to be provided with each call to the Send method. – See the first version of the Send method.
  - The last version of the Constructor does the same – establishes a default remote end-point.
- The Receive method uses the *ref* modifier to capture the IPEndPoint of the remote client.
  - This is necessary for the receiving node to communicate back to the sender as there is no connection.
  - The receiving node has to first create a dummy IPEndPoint and then use it to call the *Receive* method.
  - The Receive method will then replace the content of this dummy IPEndPoint with that of the sending node.

Algorithm for programming a UDP C/S

- Since there is no Connection, steps involved in creating a UDP server and Client are simpler than those in TCP.
- Server:
  - a) Create the Server object - e.g using UdpClient class.
  - b) Bind the Server to a specific local IPEndPoint.
  - c) Create a dummy EndPoint instance to use in capturing client's end-point
  - d) Repeat: Read a request and capture client's end-point, process the request, and send back the response.
- Client:
  - a) Create a Client object - e.g using UdpClient class.
  - b) Send a service request to the server's end-point, receive and process the response.

Example

- Echo Client/Server Application using UdpClient class
  - Again it is important do decide the **protocol**.
  - Since there is no connection, the server does not need to send any welcome message.
  - It just receives a message from any client and echo the same back to the client.



Writing a UDP Server using UdpClient

- The following is an Echo Server using the UdpClient class.

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

class SimpleUdpServer
{
   public static void Main()
   {
      IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);
      UdpClient server = new UdpClient(localEP);

      Console.WriteLine("Waiting for a client...");

     //dummy IP
     IPEndPoint remoteEP =  new IPEndPoint(IPAddress.Any, 0);
```

Writing a UDP Server using UdpClient …

```
14.         byte[] data;
15.         while(true)
16.         {
17.            data = server.Receive(ref remoteEP);
18.            Console.Write("Received from {0}: ", remoteEP.ToString());
19.            Console.WriteLine(Encoding.ASCII.GetString(data, 0,
20.                                                        data.Length));
21.            server.Send(data, data.Length, remoteEP);
22.         }
23.      }
24.  }
```

- Note the following points arising from the connection-less nature of UDP:
  - There is no ***AcceptClient*** method that returns the client's socket.
  - Thus, **NetworkStream**, **StreamReader** and **StreamWriter** cannot be used for exchange of messages.
  - The server is automatically multi-client.

Writing a UDP Client using UdpClient

- The following is an Echo Client using the UdpClient class.

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

5.  class SimpleUdpClient {
6.     public static void Main()   {
7.        UdpClient client = new UdpClient("127.0.0.1", 9050);

8.        IPEndPoint remoteEP =  new IPEndPoint(IPAddress.Any, 0);

9.        byte[] data;
10.       string input;
```

Writing a UDP Client using UdpClient …

```
        while(true)
        {
           Console.Write("Enter message for server, exit to stop: ");
           input = Console.ReadLine();
           if (input == "exit")
              break;

           client.Send(Encoding.ASCII.GetBytes(input), input.Length);
           data = client.Receive(ref remoteEP);
           Console.Write("Echo Received from {0}: ",
                                    remoteEP.ToString());
           Console.WriteLine(Encoding.ASCII.GetString(data, 0,
                                                  data.Length));
        }
        Console.WriteLine("Stopping client");
        client.Close();
     }
  }
```


Writing a UDP Server using Socket class

- The **Socket** class can also be used for creating UDP servers and clients
- Because UDP is connectionless, to create a server application:
  - Create a Socket object
  - Bind the socket to a local IPEndPoint
  - Use SendTo() and ReceiveFrom() methods to exchange messages
    - **Send** and **Receive** methods cannot be used since there is no connection

Writing a UDP Server using Socket class …

- The *SendTo* and *ReceiveFrom* methods have various overloaded versions as follows:

```
int SendTo(byte[] data, EndPoint remote);
int SendTo(byte[] data, SocketFlags flag, EndPoint remote);
int SendTo(byte[] data, int size, SocketFlags flag, EndPoint remote);
int SendTo(byte[] data, int offset, int size, SocketFlags flag,
                                                EndPoint remote);
```

```
int ReceiveFrom(byte[] data, ref EndPoint remote);
int ReceiveFrom(byte[] data, SocketFlags flag,  ref EndPoint remote);
int ReceiveFrom(byte[] data, int size, SocketFlags flag,
                                                ref EndPoint remote);
int ReceiveFrom(byte[] data, int offset, int size,  SocketFlags flag,
                                          ref EndPoint remote);
```


Writing a UDP Server using Socket class …

- The following is an Echo Server using the Socket class:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSocketServer
{
  public static void Main()
  {
    IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);

    Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Dgram, ProtocolType.Udp);

    server.Bind(localEP);
    Console.WriteLine("Waiting for a client...");

    //dummy end-point
    IPEndPoint remoteEP = new IPEndPoint(IPAddress.Any, 0);
```

Writing a UDP Server using Socket class …

```
19.      int recv;
20.      byte[] data;

21.      while(true)
22.      {
23.        data = new byte[1024];
24.        recv = server.ReceiveFrom(data, ref remoteEP);
25.        Console.Write("Received from {0}: ", remoteEP.ToString());
26.        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
27.        server.SendTo(data, recv, SocketFlags.None, remoteEP);
28.      }
29.    }
30.  }
```

- Notes:
  - Receive method requires that IPEndPoint be upcast to base type, EndPoint
  - Again **NetworkStream**, **StreamReader** and **StreamWriter** cannot be used for exchange of messages.

Writing a UDP Client using Socket class

- The following is an Echo Client using the Socket class:

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.Text;
5.  class SimpleUdpClient
6.  {
7.    public static void Main()
8.    {
9.      Socket client = new Socket(AddressFamily.InterNetwork,
10.                                 SocketType.Dgram, ProtocolType.Udp);
11.
12.     EndPoint remoteEP = new
13.             IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
14.     //Note -- need to use EndPoint for the ReceiveFrom to work!
15.
16.     byte[] data;
17.     string input;
18.     int recv;
```

Writing a UDP Client using Socket class …

```
19.      while(true)
20.      {
21.        Console.Write("Enter message for server, exit to stop: ");
22.        input = Console.ReadLine();
23.        if (input == "exit")
24.             break;
25.        client.SendTo(Encoding.ASCII.GetBytes(input), remoteEP);
26.        data = new byte[1024];
27.        recv = client.ReceiveFrom(data, ref remoteEP);
28.        Console.Write("Echo from from {0}: ", remoteEP.ToString());
29.        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
30.      }
31.      Console.WriteLine("Stopping client");
32.      client.Close();
33.    }
34.  }
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- 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