Network Programming
Network Programming

- Network Protocols
- Communication Connection
- Naming

**Why are these important?**

You are developing a multi-player game, you will need to know how to:

- Establish connections between machines
- Pass information back and forth
Computer networks

- A network is a hierarchical system of boxes and wires organized by geographical proximity
  - LAN (local area network) spans a building or campus
    - Ethernet is most prominent example
  - WAN (wide-area network) spans country or world
- An internetwork (internet) is an interconnected set of networks
Network Architecture

The Seven Layers of OSI

- Physical Layer
- Data Link Layer
- Network Layer
- Transport Layer
- Session Layer
- Presentation Layer
- Application Layer

User

Transmit Data

Receive Data

Physical Link
Layered Protocols (OSI model)
Protocols for the communication between presentation layer and user, e.g., FTP

Connection between applications, can translate data from one form to another, e.g., Encryption, e.g., SSL

Establish connection between users, session logic, e.g., sockets with TCP

Provides protocols for transportation of data packets: e.g., TCP, UDP

Provides means of transferring variable data, dealing with packets and routing addresses

Provides function and procedure for data transmission. Error correction.

Physical devices and transfer of data through buses
Physical devices and transfer of data through buses

Provides function and procedure for data transmission. Error correction.

Provides means of transferring variable data, dealing with packets and routing addresses

Establishes connection between users, session logic, e.g., sockets with TCP

Provides protocols for transportation of data packets: e.g., TCP, UDP

Provides means of transferring variable data, dealing with packets and routing addresses

Protocols for the communication between presentation layer and user, e.g., FTP

Connection between applications, can translate data from one form to another, e.g., Encryption, e.g., SSL
Ethernet segment (Layer 1)

- Ethernet segment consists of a collection of hosts connected by wires to a hub
- Spans room or floor in a building
- Operation
  - Each Ethernet adapter has a unique 48-bit address
  - Hosts send bits to any other host in chunks called frames
  - Hub slavishly copies each bit from each port to every other port
Bridged Ethernet segment

- Spans building or campus
- Bridges cleverly learn which hosts are reachable from which ports (Table of Mac addresses) and then selectively copy frames from port to port
Physical devices and transfer of data through buses

Provides function and procedure for data transmission. Error correction.

Provides means of transferring variable data, dealing with packets and routing addresses

Protocols for the communication between presentation layer and user, e.g., FTP

Connection between applications, can translate data from one form to another, e.g., Encryption, e.g., SSL

Establish connection between users, session logic, e.g., sockets with TCP

Provides protocols for transportation of data packets: e.g., TCP, UDP

Provides means of transferring variable data, dealing with packets and routing addresses

Provides function and procedure for data transmission. Error correction.

Physical devices and transfer of data through buses

**OSI MODEL**

**Application Layer**
Type of communication: E-mail, file transfer, client/server.

**Presentation Layer**
Encryption, data conversion: ASCII to EBCDIC, BCD to binary, etc.

**Session Layer**
Starts, stops session. Maintains order.

**Transport Layer**
Ensures delivery of entire file or message.

**Network Layer**
Routes data to different LANs and WANs based on network address.

**Data Link (MAC) Layer**
Transmits packets from node to node based on station address.

**Physical Layer**
Electrical signals and cabling.
## Data Link Layer

Discussion between a receiver and a sender

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data 0</td>
<td></td>
<td>A sends data message 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Data 0</td>
<td>B gets 0, sees bad checksum</td>
</tr>
</tbody>
</table>
| 2    | Data 1 | Control 0 | A sends data message 1  
                   |       |                                               | B complains about the checksum |
| 3    | Control 0 | Data 1 | Both messages arrive correctly            |
| 4    | Data 0 | Control 1 | A retransmits data message 0  
                   |       |                                               | B says: "I want 0, not 1"     |
| 5    | Control 1 | Data 0 | Both messages arrive correctly            |
| 6    | Data 0 |       | A retransmits data message 0 again        |
| 7    |       | Data 0 | B finally gets message 0                  |
### OSI Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Layer</strong></td>
<td>Type of communication: E-mail, file transfer, client/server.</td>
</tr>
<tr>
<td><strong>Presentation Layer</strong></td>
<td>Encryption, data conversion: ASCII to EBCDIC, BCD to binary, etc.</td>
</tr>
<tr>
<td><strong>Session Layer</strong></td>
<td>Starts, stops session. Maintains order.</td>
</tr>
<tr>
<td><strong>Transport Layer</strong></td>
<td>Ensures delivery of entire file or message.</td>
</tr>
<tr>
<td><strong>Network Layer</strong></td>
<td>Routes data to different LANs and WANs based on network address.</td>
</tr>
<tr>
<td><strong>Data Link (MAC) Layer</strong></td>
<td>Transmits packets from node to node based on station address.</td>
</tr>
<tr>
<td><strong>Physical Layer</strong></td>
<td>Electrical signals and cabling.</td>
</tr>
</tbody>
</table>

#### Protocols
- Protocols for the communication between presentation layer and user, e.g., FTP
- Connection between applications, can translate data from one form to another, e.g., Encryption, e.g., SSL
- Establish connection between users, session logic, e.g., sockets with TCP
- Provides protocols for transportation of data packets: e.g., TCP, UDP
- Provides means of transferring variable data, dealing with packets and routing addresses
- Provides function and procedure for data transmission. Error correction.

Physical devices and transfer of data through buses
Routers (Layer 3, Network Layer)

- Multiple incompatible LANs can be physically connected by specialized computers called routers.
- The connected networks are called an internet.

LAN 1 and LAN 2 might be completely different, totally incompatible LANs (e.g., Ethernet and ATM).
Internet Protocol

How is it possible to send bits across incompatible LANs and WANs?

Protocol Software

- smooth out the differences between the different networks
- Internet Protocol (i.e., set of rules) governs how hosts and routers should cooperate when they transfer data from network to network
Solving two issues:

- **Naming scheme**
  - The internet protocol defines a uniform format for host addresses
  - Each host (and router) is assigned 1+ internet addresses that uniquely identifies it

- **Delivery mechanism**
  - The internet protocol defines a standard transfer unit (*packet*)
  - Packet consists of *header* (including packet-size, source and destination) and *data*
Transferring data over internet

1. Data (client) to protocol software
2. Internet packet (LAN1 frame)
3. Data (LAN1 adapter)
4. Data (LAN1 adapter)
5. Data (LAN2 adapter)
6. Data (LAN2 adapter)
7. Data (LAN2 adapter)
8. Data (server)
Communication Protocols

- **UDP (User Datagram Protocol)**
  - Connectionless
  - No congestion control
  - No ordering of packets
  - unreliable

- **TCP (Transmission Control Protocol)**
  - Handshaking, Congestion control
  - Orders packages
  - reliable
# Layered Protocols (OSI model)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.</td>
<td>FTP, Telnet, SMTP, X400, X500</td>
</tr>
<tr>
<td><strong>Presentation</strong></td>
<td>Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.</td>
<td>XDR, ASN.1, encryption</td>
</tr>
<tr>
<td><strong>Session</strong></td>
<td>At this level communication between processes is established and error recovery is performed. It is not required for connectionless communication.</td>
<td></td>
</tr>
<tr>
<td><strong>Transport</strong></td>
<td>This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports, Protocols in this layer may be connection-oriented or connectionless.</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through PSEs or routers. In a single LAN no routing is required.</td>
<td>X25, IP</td>
</tr>
<tr>
<td><strong>Data link</strong></td>
<td>Responsible for error-free transmission of packets between computers that are directly connected. In a WAN the connections are between pairs of PSEs and PSEs and hosts. In a LAN they are pairs of hosts.</td>
<td>HDLC, Ethernet: CSMA/CD</td>
</tr>
<tr>
<td><strong>Physical</strong></td>
<td>The circuits and hardware that drives the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre-optic circuits) or electromagnetic signals (on radio and microwave circuits).</td>
<td>X.21, Ethernet: baseband signalling</td>
</tr>
</tbody>
</table>
Other issues

- We are glossing over a number of important questions:
  - What if different networks have different maximum frame sizes? (segmentation)
  - How do routers know where to forward frames?
  - How are routers informed when the network topology changes?
  - What if packets get lost?

- These questions form the heart of the area of computer systems known as networking
IP Addresses

32-bit IP addresses are stored in an IP address struct

```c
/* Internet address structure */
Class System.Net.IPAddress
{
    many properties including
    long Address;
    System.Net.AddressFamily AddressFamily;
}
```
Internet Domain Names

first-level domain names
  \[ \text{mil} \quad \text{edu} \quad \text{gov} \quad \text{com} \]

second-level domain names
  \[ \text{.mit} \quad \text{northwestern} \quad \text{gatech} \quad \text{amazon} \]

third-level domain names
  \[ \text{cs} \quad \text{ece} \quad \text{www} \quad 208.216.181.15 \]

unlimited root

Address:
  - godzilla: 129.105.99.240
  - jumpstart: 129.105.44.45
Domain Naming System (DNS)

- The Internet maintains mapping IP-addr/domain names in a distributed database - DNS

- Conceptually - DNS database as being millions of host entry structures

System.Net.Dns

```csharp
public static IPHostEntry GetHostByName(string hostName);
public static IPHostEntry GetHostByAddress(string address);
public static string GetHostName();
public static IPHostEntry Resolve(string hostName);
```
Domain Naming System (DNS)

String name = “neu.edu”;
IPHostEntry iphost = Dns.Resolve(name);
IpAddress ip = iphost.AddressList[0];
Anatomy of an Internet connection

**client socket address**
128.2.194.242:51213

**connection socket pair**
(128.2.194.242 :51213, 208.216.181.15:80)

**server socket address**
208.216.181.15:80

**client host address**
128.2.194.242

**server host address**
208.216.181.15
Domain Naming System (DNS)

String name = “localhost”;

IPHostEntry iphost = Dns.Resolve(name);

IpAddress ip = iphost.AddressList[0];

int portNumber = 3030;

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

IPEndPoint Endpoint = new IPEndPoint (ip,
    System.Convert.ToInt16(portNumber));

s.Connect (Endpoint);
A client-server transaction

- A network application based on the client-server model:
  - a server process and one or more client processes
  - server manages some resource
  - server provides service by manipulating resource for clients

Note: clients and servers are processes running on hosts (can be the same or different hosts).
Socket Programming

Blocking Sockets Code
Client Side

- Trying to connect to a particular server

- What do we know?
  - IP address
  - Port that we know the server is listening at
  - Type of data it is accepting
Client side code: connecting to server

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

// formatting the ip address + port
String ip = “127.0.0.1”;
String port = “8221”;
int myPort = System.Convert.ToInt16(port, 10);
System.Net.IPAddress ipaddr = System.Net.IPAddress.Parse(ip);

// making the end point and connecting to it
System.Net.IPEndPoint connPt = new System.Net.IPEndPoint(ipaddr, myPort);
s.Connect(connPt);
Client side code: send a message

```
try
{
    // correct byte format

    Object someData = SomeMessage;

    byte [] byData = System.Text.Encoding.ASCII.GetBytes(someData.ToString());

    // send the bytes over through the socket s

    s.Send(byData);
}
```

Catch (System.Net.Sockets.SocketException se) {}
try
{
    // get the message
    byte [] buffer = new byte [1024];
    int lengthbuff = s.Receive (buffer);

    // format the bytes into a string
    char chars = new char[lengthbuff];
    int charLen = d.GetChars (buffer, 0, lengthbuff, chars, 0);
    System.String data = new System.String(chars);

    // print data
}

Catch (System.Net.Sockets.SocketException se) {}
Server Side

- Opens a port and listens for connections

- Requirement:
  - We want to listen for messages coming from any IP address
  - Send the message back
  - Prints the message on server screen
Server side code: Listen thru a socket

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

System.Net.IPEndPoint **connPt** = new System.Net.IPEndPoint(System.Net.IPAddress.Any, 8221);

s.**Bind**(connPt);

s.**Listen**(4); // 4 is no. of connections
Server Side Code: Accepting a connection

```
Socket handler = s.Accept();  // synchronous call

// this handler is the one that you will use to send and receive
```
Server Side Code : receiving

String data = "";

Byte [] bytes = new byte[1024];

int NoBytesRec = handler.Receive(bytes);

data = Encoding.ASCII.GetString(bytes, 0, NoBytesRec);

handler.Send(bytes);  // sending message

Console.WriteLine(data);  // writing message
Server Side

Blocking Sockets
```csharp
using System;
using System.Threading; // For threads...
using System.Net; // For networking fun!
using System.Net.Sockets; // For sockets fun!
using System.Text; // For easy conversion!

namespace MyServer
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class MyServer
    {
        // The socket
        Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        // IPEndPoint for connection
        IPEndPoint client = new IPEndPoint(IPAddress.Any, 2005); // Fixed port

        // Array of wonderful handlers
        Socket[] workers = new Socket[10];
        // Array of threads
        Thread[] connections = new Thread[10];
        int count = 0; // Count! Used for keeping track of clients

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        /// < STAThread>
        [STAThread]
        static void Main(string[] args)
        {
            MyServer server = new MyServer(); // Instantiate an instance of our the server
            server.mySocket.Bind(server.client); // Bind to the endpoint that will be...the client

            // A little initial status
            Console.WriteLine("MyServer started. Now listening on port: {0}\n", server.client.Port);
        }
    }
}
// Here is our endless loop!
while(true)
{
    server.waitForBuddies(); // Call our receiving function
} // Keep listening after every connection

private void waitForBuddies()
{
    mySocket.Listen(10); // Accept up to 10 clients
    workers[count] = mySocket.Accept(); // Assign a handler
    connections[count] = new Thread(new ThreadStart(letsTalk));
    connections[count].Name = count.ToString(); // Need to know which one it is!
    connections[count].Start();

    Console.WriteLine("Just connected to client #{0} at address {1}", count+1, workers[count].RemoteEndPoint.ToString());

    count++; // Increase our position
    //Console.WriteLine(count);
}
private void letsTalk()
{
    byte[] theMessage = new byte[1024]; // The data to be received!
    int currentClient = System.Int32.Parse(Thread.CurrentThread.Name); // The worker/client we are with

    // While messages are still "real", keep accepting/listening!
    while(this.printMessage(workers[currentClient], theMessage))
    {
        Console.WriteLine("Client Disconnected");
        count--;
    }
}

private bool printMessage(Socket s, byte[] b)
{
    int bytesReceived = s.Receive(b); // Receive the data
    if(bytesReceived > 0) // If the message is "real"
    {
        string message = System.Text.Encoding.ASCII.GetString(b,0,bytesReceived);
        Console.WriteLine(message); // Write the string!
        s.Send(b); // Send it back...
        return true;
    }

    return false; // The bytes were 0, disconnect!
}
Client Side

Blocking Sockets
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net; // For network fun!
using System.Net.Sockets; // For socket fun!
using System.Text; // For easy conversion!
using System.Threading; // AHHHH
	namespace MyClient
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
	nothing
	public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.RichTextBox echoBox;
    private System.Windows.Forms.TextBox messageBox;
    private System.Windows.Forms.Button btnSend;

    // The socket!
    Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    // The IPEndPoint
    // Can change the address to whatever IP you want...
    IPEndPoint dest = new IPEndPoint(IPAddress.Loopback, 2005); // Local machine and fixed port for now

    /// <summary>
    /// Required designer variable.
    /// </summary>
    /// <summary>
    private System.ComponentModel.IContainer components = null;

    public Form1()
    {
        // Your code here
    }
}
private byte[] toBytes(string s)
{
    return(Encoding.ASCII.GetBytes(s)); // Converts our string to bytes and returns it!
}

private void Form1_Activated(object sender, System.EventArgs e)
{
    // Just some initialization...and connecting!
    echoBox.AppendText("Trying to Connect...");
    mySocket.Connect(dest); // Connect to the server!
    echoBox.AppendText("Connected!\n");
}

private void echoServer()
{
    byte[] message = new byte[1024];
    mySocket.Receive(message); // Block and wait for the server response
    // Display it
    echoBox.AppendText(Encoding.ASCII.GetString(message, 0, message.Length));
    echoBox.AppendText("\n");
}

private void btnSend_Click(object sender, System.EventArgs e)
{
    // If the textbox actually HAS something, send it and echo!
    if (!messageBox.Text.Equals"")
    {
        mySocket.Send(this.toBytes(messageBox.Text));
        echoServer();
    }
}
Class Assignment

Use the socket library to connect several clients to a server. Build the clients with a textbox interface where clients can type in string messages. The server should just echo the string on its own console interface and then sends it to the Client. Use blocking socket calls for this assignment.
Delegates

- Pointer to a method
  - Name
  - Arguments
  - Return type
private delegate void printme (string message);
public static void printmsginbox (string msg)
{
    MessageBox.Show(msg);
}

static void Main(string[] args)
{
    printme printinghello = new printme
    (printmsginbox);
    printinghello ("Hello World");
}
Delegate Example

```csharp
private delegate void printme(string message);
public static void printmsg1(string msg)
{
    MessageBox.Show(msg);
}
public static void printmsg2(string msg)
{
    Console.WriteLine(msg);
}
static void Main (string[] args)
{
    printme printinghello = new printme (printmsg1);
    printinghello += new printme(printmsg2);
}
```
Asynchronous Calls

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

System.Net.IPEndPoint connPt = new System.Net.IPEndPoint(System.Net.IPAddress.Any, 8221);

s.Bind (connPt);

s.Listen (4); // 4 is no. of connections

s.BeginAccept (new AsyncCallback(OnConnect), s);

public delegate void AsyncCallback(IAsyncResult ar);
Asynchronous Calls

```csharp
s.BeginAccept (new AsyncCallback(OnConnect), s);

public void OnConnectRequest( IAsyncResult ar )
{
    Socket listener = (Socket) ar.AsyncState; //state passed in
    client = listener.EndAccept( ar ); //handler
    Console.WriteLine( client.RemoteEndPoint + “ joined” );

    client.BeginReceive (bytes,0,bytes.Length , SocketFlags.None, new AsyncCallback(receiveData), client);

    listener.BeginAccept( new AsyncCallback(OnConnect), listener );
}
```
Asynchronous Calls

```csharp
client.BeginReceive(bytes, 0, bytes.Length, SocketFlags.None, new AsyncCallback(receiveData), client);

public void receiveData (IAsyncResult ar)
{
    Socket s = (Socket) ar.AsyncState;
    try{
        int nBytesRec = s.EndReceive(ar);
        string sReceived = Encoding.ASCII.GetString(bytes);
        Console.WriteLine("message received is " + sReceived + ":");
        s.Send(bytes);
        s.BeginReceive(bytes, 0, bytes.Length, SocketFlags.None, new AsyncCallback(receiveData), s);
    }
}
```
Server Side
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net.Sockets;
using System.Threading;
using System.Text;

namespace ServerAssign4
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.RichTextBox receivedMessageBox;
        private Socket mySocket;
        private Socket client;
        private System.Threading.Thread connectionThread;
        private byte[] bytes = new byte[1024];
        private System.ComponentModel.IContainer components;

        public delegate void AddMessageBoxDel(string item);

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```csharp
static void Main()
{
    Application.Run(new Form1());
}

private void startButton_Click(object sender, System.EventArgs e)
{
    connectionThread = new Thread(new System.Threading.ThreadStart(Connections));
    connectionThread.Start();
}

private void Connections()
{
    try
    {
        mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
    //connecting to any ip address on port 8822
    System.Net.IPEndPoint connPt = new System.Net.IPEndPoint(System.Net.IPAddress.Any, 8822);
        mySocket.Bind(connPt); //binding address
        mySocket.Listen(10); //listening for up to 10 connections
        mySocket.BeginAccept(new AsyncCallback(OnConnect), mySocket);
    }
    catch (System.Net.Sockets.SocketException se)
    {
    }
}

private void OnConnect(IAsyncResult ar)
{
    Socket listener = (Socket)ar.AsyncState; //state passed in;
    client = listener.EndAccept( ar );
    client.BeginReceive(bytes, 0, bytes.Length, SocketFlags.None, new AsyncCallback (receiveData), client); //asynch call to accepting connections again
    listener.BeginAccept(new AsyncCallback( OnConnect ), listener );
}
private void receiveData (IAsyncResult ar)
{
    Socket myClient = (Socket)ar.AsyncState; //state passed in;
    try
    {
        //Check Connection Status and remove if not connected to make room for more users
        if (!myClient.Connected)
        {
            myClient.Shutdown(System.Net.Sockets.SocketShutdown.Both);
            myClient.Close();
        }
        int nBytesRec = myClient.EndReceive( ar);
        byte[] localbytes = bytes;
        bytes = new byte[1024];
        string clientReceived = Encoding.ASCII.GetString(localbytes);
        if (this.receivedMessageBox.InvokeRequired)
        {
            AddMessageBoxDel d = new AddMessageBoxDel(PutText);
            this.Invoke(d, new object[] {"\n" + clientReceived + "\n"});
        }
        else
        {
            this.receivedMessageBox.AppendText(" The following information was received: " + clientReceived + " : " + "\n"");
        }
        myClient.Send(localbytes);
        //asynch call to receive again
        myClient.BeginReceive( bytes, 0, bytes.Length, SocketFlags.None, new AsyncCallback(receiveData), myClient);
    }
    catch (System.Net.Sockets.SocketException se)
    {
        if (!myClient.Connected)
        {
            myClient.Shutdown(System.Net.Sockets.SocketShutdown.Both);
            myClient.Close();
        }
    }
}
private void PutText(string text)
{
    receivedMessageBox.AppendText(text);
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
}

private void Form1_Load(object sender, System.EventArgs e)
{
}

private void notifyIcon1_MouseDown(object sender, System.Windows.Forms.MouseEventHandler e)
{
}

private void textBox1_TextChanged(object sender, System.EventArgs e)
{
}
Client Side
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net.Sockets;
using System.Text;

namespace ClientAssign4
{
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TextBox sendMessageBox;
        private System.Windows.Forms.RichTextBox receiveMessageBox;
        private System.Windows.Forms.Label receiveMessageLabel;
        private Socket s;
        private byte[] buffer = new byte[1024];

        private System.ComponentModel.Container components = null;
        public delegate string AddMessageInBoxDel();
        public delegate void AddMessageOutBoxDel(string item);
public Form1()
{
    InitializeComponent();

    s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
    string ip = "127.0.0.1";
    int port = Convert.ToInt16(8822);
    System.Net.IPEndPoint connPt = new System.Net.IPEndPoint(ipAddy, port);
    try
    {
        s.Blocking = false;
        AsyncCallback connecting = new AsyncCallback(OnConnect);
        s.BeginConnect(connPt, connecting, s);
    }
    catch (System.Net.Sockets.SocketException se)
    {
    }
}
public void OnConnect( IAsyncResult ar )
{
    // Socket comes in
    Socket myServer = (Socket)ar.AsyncState;
    try
    {
        afterConnect(myServer);
    }
    catch( Exception se )
    {
        MessageBox.Show( this, se.Message, "Failed to connect due to error" );
    }
}

public void afterConnect ( Socket myServer)
{
    if (myServer.Connected)
    {
        AsyncCallback recieveData = new AsyncCallback(OnRecievedData);
        myServer.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None, recieveData, myServer);
    }
    else
    {
    }
}
public void OnRecievedData ( IAsyncResult ar )
{
    Socket myServer = (Socket)ar.AsyncState;
    try
    {
        int bufferlength = myServer.EndReceive( ar );
        if( bufferlength > 0 )
        {
            string inMessage = Encoding.ASCII.GetString( buffer, 0, bufferlength );
            if (this.receiveMessageBox.InvokeRequired)
            {
                AddMessageOutBoxDel d = new AddMessageOutBoxDel(PutText);
                this.Invoke(d, new object[] { inMessage });
            }
            else
            {
                this.receiveMessageBox.AppendText(inMessage + "\n");
            }
            afterConnect(myServer);
        }
    }
    catch( Exception se )
    {
        MessageBox.Show( this, se.Message, "Unusual error during Recieving" );
    }
}

private void PutText(string text)
{
    receiveMessageBox.AppendText(text);
}

private string GetText()
{
    return (sendMessageBox.Text);
}
private void sendButton_Click(object sender, System.EventArgs e)
{
    string theMessage;
    if (this.sendMessageBox.InvokeRequired)
    {
        AddMessageInBoxDel d = new AddMessageInBoxDel(GetText);
        theMessage = this.Invoke(d).ToString();
    }
    else
    {
        theMessage = sendMessageBox.Text;
        sendMessageBox.Text = "";
    }
    Object someData = theMessage;
    // send message
    try
    {
        byte[] byData = System.Text.Encoding.ASCII.GetBytes(someData.ToString());
        s.Send(byData);
    }
    catch (System.Net.Sockets.SocketException se)
    {
    }
}
Class Assignment

Use the socket library to connect several clients to a server. Build the clients with a textbox interface where clients can type in string messages. The server should just echo the string on its own console interface and then sends it to all Client. Use asynchronous socket calls for this assignment.
So what happens when you want to send objects through sockets?