The Internet: Networking with Stream-based Sockets

The Internet

- A Global Network of Networks
  - Defense Dept. Advanced Research Projects Agency (DARPA)
  - Stanford Research Institute (Doug Engelbart)
  - Designed to survive bomb attacks
  - Distributed control, Expandable
- Ethernet
  - Global standard for interconnecting computers
  - Xerox PARC (Early 70s)
  - Client/Server architecture
- Exponential Growth
  - Tens of Millions of Computers
  - Hundreds of millions of Users
The Internet

- A Packet Switched Network
  - Like Postal System
  - Messages broken up into packets (like envelopes)

<table>
<thead>
<tr>
<th>Error Detect</th>
<th>Data</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Check Sum)</td>
<td></td>
<td>(Addresses)</td>
</tr>
</tbody>
</table>

Computer Node Addresses:

- IP (Internet Protocol)
  - 32 bit numeric address in four 8-bit fields:
  - 128.226.6.4 (bingsuns IP Address)
    - network computer
    - (city/state) (street/number) <-- postal analogy
  - Called the IP Address

- TCP (Transmission Control Protocol):
- **Send Site**: Breaks message into packets
- **Receive Site**: Collects & Reassembles packets in proper order
“Best” path between computers is chosen using **Routers**

![Diagram]

**Domain Names**

- Synonyms for IP Addresses
- `bingsuns.binghamton.edu`
  - individual
  - largest
  - machine
  - domain
  - Synonym for 128.226.6.4
- Internet Domain Name Server (DNS) software maps domain names to IP addresses
Common High-Level Domain Names

- com: commercial
- edu: educational
- gov: government
- mil: military
- org: other organization
- net: network resources
- --: country name
  - e.g., ca = Canada

The .NET Dns Class

- In System.Net namespace
- **Dns**: a class that has methods that retrieve information about a specific host from the Domain Name Server
  - Dns.GetHostByDomain(string hostName) and Dns.GetHostByAddress(string hostIPaddress) static methods
  - Both return an IPEndPoint object containing host information
    - For GetHostByName(hostName) it gives access to the IP address(es) corresponding to the DNS name specified in `hostName`
      - That object’s AddressList property can be used to set up an array of IP addresses that correspond to the hostname
    - For GetHostByAddress(hostIPaddr) it gives access to domain name/aliases for the specified IP address
      - That object’s Aliases property is an array of domain names
  - See GetIPAddress example program
Networking Software

- **Client/Server Model**
  - Client Program -- seeks a service from remote computer
  - Server Program -- provides a service to a client running on a remote computer
  - Computers are usually connected over a network
  - Examples
    - Print Server
    - File Server
    - Information Server
Information Servers

• Program handles requests for information
• Some examples
  – e-mail: electronic mail service
  – telnet/Rlogin/SSH: remote logon services
  – ftp/SSH: file transfer service
  – Some older text-based information servers:
    • gopher: net browsing service (text based)
    • archie/veronica: automated net search services
    • WAIS: automated file content search service
    • Net News: network bulletin board service
  – WWW: hypermedia access to internet (Web page service)

Network Communication Between Computers

• Applications running on different computers can communicate with each other
  – Server Application: Waits for other applications on other computers to open a communication connection
  – Client Application: Attempts to open a connection
• When connection is established, data can be exchanged
• Either can close the communication
• Connections:
  – Two programs running on different computers that are communicating with each other form a connection
  – Data is sent and received along the connection
**Network Socket Stream**

- Basic object used to perform network communication
- Used to read/write messages going between apps
  - (Like a file stream in file I/O)
- A **Socket** is a communication "endpoint“
  - There's a socket at each end of the connection
- Windows support for sockets: in the **Winsock API**
  - MFC encapsulates this in the **CAsyncSocket** base class
    - Provides complete, event-driven socket communications
    - Lowest level support  -- Notes at: www.cs.binghamton.edu/~reckert/360/17b_sockets_f03.html
    - Higher level support from derived classes like **CSocket**
- .NET encapsulates socket support in:
  - **System.Net.Sockets** namespace
  - With .NET sockets, networking is viewed like file I/O
    - Read from /write to a socket stream as easily as from/to a file stream

**Making a Socket Connection to a Process Running on Another Computer**

- Specify the **IP Address** of computer where the other application is running
  - Identifies a machine
- Also specify the **Port** the application is listening on
  - Identifies the program that should handle the communication
    - e.g. port 80 is reserved for web document transfer
- IP Address/Port are like number/extension in telephone communication
  - Port can be any number from 0 to 65535
    - Numbers 0 to 1023 may be used by the operating system
    - So use numbers greater than 1023
Details of Establishing a Simple Server (Using TCP/IP Network Socket Streams)

1. Create a `TcpListener` class object
   - `TcpListener myListener = new TcpListener(5000);`
   - Parameter: port # to bind the Server to on the machine it’s running on

2. Call `TcpListener` object’s `Start()` method to start listening for connection requests
   - `myListener.Start();`

3. Use `TcpListener`’s `AcceptSocket()` to accept an incoming request and establish the connection
   - `Socket myConnection = myListener.AcceptSocket();`
   - Returns a `Socket` object
     - Socket object will be null if connection was not made
     - Its `Connected` property will be true after socket is connected

4. Create a `NetworkStream` associated with the socket
   - `NetworkStream myNetStream = new NetworkStream(myConnection);`
   - This will be used to do the reading and writing as in File I/O

Using the Server Network Stream Connection

5. Create `BinaryReader` and `BinaryWriter` objects for transferring data across the network stream
   - `BinaryWriter myWriter = new BinaryWriter(myNetStream);`
   - `BinaryReader myReader = new BinaryReader(myNetStream);`

6. Use `BinaryReader/BinaryWriter` methods to read/write data, e.g.:
   - `string receiveStr, sendStr;`
   - `receiveStr = myReader.ReadString();`
     - Reads a line of text from the network stream (sent by the Client)
   - `myWriter.Write(sendStr);`
     - Writes the specified string to the network stream (to the Client)

7. When done, close readers, writers, network stream, and connection socket
   - `myReader.Close(); myWriter.Close();`
   - `myNetStream.Close(); myConnection.Close();`
Details of Establishing a Simple Client (Using Network Streams)

1. Create a **TcpClient** class object
   
   ```csharp
tcpClient myClient = new tcpClient();
   ```

2. Try to connect to a Server
   - Call Client object’s Connect(IP address, port) method
     - Specify IP address (or domain name) of machine Server is running on and Server’s port number in the two parameters
     - If successful, an underlying socket will be created for communications and a positive integer is returned
     - Will throw an exception if no Server is available at that address & port
     ```csharp
     myClient.Connect(“localhost”, 5000);
     ```
     - “localhost” = “loopback” = 127.0.0.1 means same machine as server

3. Get a **NetworkStream** associated with the TcpClient
   ```csharp
   networkStream myNetStream = myClient.GetStream();
   ```
   - This will be used to do the reading and writing as in File I/O
   - An underlying socket will be created

Using the **Client** Network Stream Connection

4. Create **BinaryReader** and **BinaryWriter** objects for transferring data across the network stream
   ```csharp
   BinaryWriter myWriter = new BinaryWriter(myNetStream);
   BinaryReader myReader = new BinaryReader(myNetStream);
   ```

5. Use BinaryReader/BinaryWriter objects to read/write data
   ```csharp
   string receiveStr, sendStr;
   receiveStr = myReader.ReadString();
   ```
   - Reads a line of text from the network stream (sent by the Server)
   ```csharp
   myWriter.Write(sendStr);
   ```
   - Writes the specified string to the network stream (to the Server)

6. When done, close readers, writers, network stream, and TCP Client
   ```csharp
   myReader.Close(); myWriter.Close();
   myNetStream.Close(); myClient.Close();
   ```
Using Threads with Sockets

• Whenever we try to establish and use a connection, the thread we do it in blocks until the connection is established
  – Blocking also takes place when reading or writing data
• To avoid the entire application from freezing, run this code in a separate thread

A Network Chat Client/Server System

• A Server and a Client Application
  – See Chapter 23 in your Deitel text book
• ChatServer application waits for a Client application to connect to a specified port on its computer
• ChatClient application attempts to connect to that port on that machine
• Both ChatServer and ChatClient have a single-line “input” text box and a multi-line “display” text box
• When a connection is established, either can type text in its input text box and the text will appear in the other’s display text box when user hits <Enter> key
• The communication is done through network streams
### ChatServer Application

- Form’s constructor starts a new thread to accept Client connections
  - Thread’s RunServer( ) method does the work (executes when thread starts)
  - Creates and starts a TcpListener on port 5000
  - Listens for a connection attempt from a Client
    - Connection is made (socket obtained) with listener’s AcceptSocket() method
    - Uses new socket’s NetworkStream( ) method to get a network stream
    - Creates binary reader & writer to read/write data over the network stream connection
    - Enters into a do/while loop that continually uses the binary reader to read a string from the network stream
      - Any string read is added to the text displayed in the “display” text box
      - Do/While loop continues until the socket is disconnected or a “>>CLIENT TERMINATE” string is received
    - After do/while loop exits, the reader, writer, network stream, and socket are closed

- Input text box’s KeyDown handler:
  - Writes the text in the input text box to the network stream using its binary writer whenever the user types <Enter> as long as the connection is valid
  - If the text entered is “TERMINATE”, closes the connection socket

- An event handler for the form’s “Closing” event is added
  - Calls System.Environment.Exit(System.Environment.ExitCode) to close the app

- Exit( ) method of Environment class closes all threads associated with the app

### ChatClient Application

- Same overall structure as the ChatServer
- Form’s constructor starts a new thread to connect to the Server
  - Thread’s RunServer( ) method does the work
    - Instantiates a TcpClient and run its Connect(“localhost”, 5000) method
      - Connects to the Server on the same machine
      - This call blocks until connection request is accepted
    - Uses TcpClient’s GetStream( ) method to create a network stream
    - Creates a binary reader and a binary writer to read/write data over the network stream connection
    - Enters into a do/while loop that continually uses the binary reader to read a string from the network stream and display it in the form’s “Display” text box
    - After do/while loop exits, the reader, writer, NetworkStream, and TcpClient are all closed and application is closed using the Application.Exit( ) method

- Input text box’s KeyDown handler
  - Write the text in the input box to the network stream using its binary writer as in the ChatServer application

- For both the Server and the Client, it would be much better to use Try/Catch blocks